# Efficient and Scalable Simulations of Active Hydrodynamics in Three Dimensions

A dissertation submitted to
TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT INFORMATIK

to attain the degree of

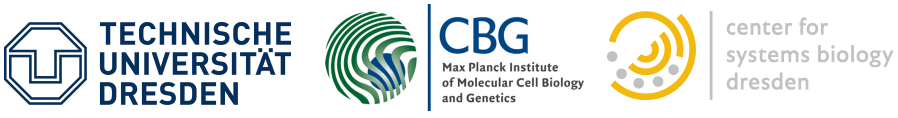
Doctor of Philosophy
(Ph.D.)

presented by

Abhinav Singh

*Master of Science*

born on 07.10.1995 in Bhusawal, India




Technische Universität Dresden
Max Planck Institute of Molecular Cell Biology and Genetics
Center for Systems Biology Dresden

Dresden, 2023

**Day of submission**: 10.08.2023
**Day of defense**: 27.11.2023

**Examination committee**:
Prof. Dr. Jeronimo Castrillon (chair of the committee)
Technische Universität Dresden, Dresden, Germany
Prof. Dr. Ivo F. Sbalzarini (reviewer)
Technische Universität Dresden, Dresden, Germany
Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany
Center for Systems Biology Dresden, Dresden, Germany
Prof. Dr. Michael Shelley (reviewer)
The Courant Institute of Mathematical Sciences, New York City, USA.
Center for Computational Biology, Flatiron Institute, New York City, USA.
Prof. Dr. Frank Jülicher
Max Planck Institute for the Physics of Complex Systems, Dresden, Germany
Prof. Dr. Bjoern Andres (Fachreferent)
Technische Universität Dresden, Dresden, Germany

*"Science knows no country, because knowledge belongs to humanity, and is the torch which illuminates the world."*

— Louis Pasteur

**Abstract**

Active matter represents a unique class of non-equilibrium systems, including examples ranging from cellular structures to large-scale biological tissues. These systems exhibit intriguing spatiotemporal dynamics, driven by the constituent particles' continuous energy expenditure. Such active-matter systems, featuring complex hydrodynamics, are described by sophisticated mathematical models, typically using partial differential equations (PDEs). PDEs modeling hydrodynamics, such as the Navier-Stokes equations, are analytically intractable, and notoriously challenging to study computationally. The challenges include the need for consistent numerical methods along with their efficient and scalable high-performance computer implementation to solve the PDEs numerically. However, when considering new theoretical PDE models, such as active hydrodynamics, conventional approaches often fall short due to the specialization made in the numerical methods to study certain specific models. The inherent complexity and nonlinearity of active-matter PDEs add to the challenge. Hence, the computational study of such active-matter PDE models requires rapidly evolving high-performance computer software that can easily implement new numerical methods to solve these equations in biologically realistic three-dimensional domains. This presents a rich, yet underexplored territory demanding scalable computational frameworks that apply to a large class of PDEs.

In this thesis, we introduce a computational framework that effectively allows for using multiple numerical methods through a context-aware template expression system akin to an embedded domain-specific language. This framework primarily aims at solving lengthy PDEs associated with active hydrodynamics in complex domains, while experimenting with new numerical methods. Existing PDE-solving codes often lack this flexibility, as they are closely tied to a PDE and domain geometry that rely on a specific numerical method. We overcome these limitations by using an object-oriented implementation design, and show experiments with adaptive and numerically consistent particle-based approach called Discretization-Corrected Particle Strength Exchange (DC-PSE). DC-PSE allows for the higher-order discretization of differential operators on arbitrary particle distributions leading to the possibility of solving active hydrodynamic PDEs in complex domains. However, the curse of dimensionality makes it difficult to numerically solve three-dimensional equations on single-core architectures and warrants the use of parallel and distributed computers.

We design a novel template-expression system and implement it in the scalable scientific computing library OpenFPM. Our methodology offers an expression-based embedded language, enabling PDE codes to be written in a form that closely mirrors mathematical notation. Leveraging OpenFPM, this approach also ensures parallel scalability. To further enhance our framework's versatility, we employ a *separation-of-concerns* abstraction, segregating the model equations from numerics, and domain geometry. This allows for the rapid rewriting of codes for agile numerical experiments across different model equations in

various geometries. Supplementing this framework, we develop a distributed algebra system compatible with OpenFPM and Boost Odeint. This algebra system opens avenues for a multitude of explicit adaptive time-integration schemes, which can be selected by modifying a single line of code while maintaining parallel scalability.

Motivated by symmetry-preserving theories of active hydrodynamics, and as a first benchmark of our template-expression system, we present a high-order numerically convergent scheme to study active polar fluids in arbitrary three-dimensional domains. We derive analytical solutions in simple Cartesian geometries and use them to show the numerical convergence of our algorithm. Further, we showcase the scalability of the computer code written using our expression system on distributed computing systems. To cater to the need for solving PDEs on curved surfaces, we present a novel meshfree numerical scheme, the Surface DC-PSE method. Upon implementation in our scalable framework, we benchmark Surface DC-PSE for both explicit and implicit Laplace-Beltrami operators and show applications to computing mean and Gauss curvature.

Finally, we apply our computational framework to exploring the three-dimensional active hydrodynamics of biological flowing matter, a prominent model system to study the active dynamics of cytoskeletal networks, celluar migration, and tissue mechanics. Our software framework effectively tackles the challenges associated to numerically solving such non-equilibrium spatiotemporal PDEs. We perform linear perturbation analysis of the three-dimensional Ericksen-Leslie model and find an analytical expression for the critical active potential or, equivalently, a critical length of the system above which a spontaneous flow transition occurs. This spontaneous flow transition is a first realization of a three-dimensional active Fréedericksz transition. With our expression system, we successfully simulate 3D active fluids, finding phases of spontaneous flow transitions, traveling waves, and spatiotemporal chaos with increasing active stress. We numerically find a topological phase transition similar to the Berezinskii–Kosterlitz–Thouless transition (BKT transition) of the two-dimensional XY model that occurs in active polar fluids after the spontaneous flow transition.

We then proceed to non-Cartesian geometries and show the application of our software framework to solve the active polar fluid equations in spherical domains. We find spontaneous flows in agreement with recent experimental observations. We further showcase the framework to solve the equations in 3D annular domains and a 'peanut' geometry that resembles a dividing cell. Our simulations further recapitulate the actin flows observed in *Xenopus* egg extracts within spherical shell geometries, showcasing our framework's versatility in handling complex geometrical modifications of model equations.

Looking ahead, we hope our framework will serve as a foundation for further advancements in computational morphogenesis, fostering collaboration and using the present techniques in biophysical modeling.

## Acknowledgments

First and foremost, I owe my deepest gratitude to Ivo. Entrusting me the freedom to dive into an interdisciplinary topic that encapsulates all the topics I truly love, combining computers, physics, and math with biology. Working with you was nothing short of a privilege. Under your guidance, I found direction and resources. This thesis, in its current form, would merely have been a dream without your unwavering support. I would also like to thank my advisory committee, Prof. Frank Jülicher, Dr. Carl Modes and Prof. Axel Voigt for helpful feedback.

Pietro, you have been a guiding star in both my academic journey and personal life. You have been a beacon of knowledge and a companion in exploring our shared passions, from the intricacies of computer hardware to the exhilaration of gaming or highly optimized software. Your friendship, unwavering support and heartfelt compassion have left an unforgettable mark on my heart. I've learned so much from you, and I am thankful for the times we have spent together.

Quentin, you always brought joy and curiosity into everyone's life. I've learned a lot about physics from you, and I will always remember our fun talks about video games and music. Thanks to Anastasia, Surya, Tina, Sachin and Nandu for the wonderful lunchtime chats and discussions. Justina, and Hannes for helping me understand German better and for all the advice. Joel, I will always remember our skiing trip and coffee chats. Juan, thanks for the great drink parties. Lenny, thanks for all our talks about interpolation, and Aryaman, our trips to Bonn were always fun. Susann and Lucie, without you navigating the German bureaucracy would have been impossible. Charlie, and Salman, thank you for being a great help during our collaboration.

I would like to especially thank Ale for being a great friend and supplementing me with new and interesting problems, always having great discussions on scientific problems, you are a great friend and colleague, and thank you for being there. Similarly, Philipp, your collaboration was the lifeline of this thesis. Our journey together has been both enlightening and gratifying. Also, a big thanks to Landfried, Serhii, Rushi, Tim, and Birte. Each of you made this work better and I hope you enjoyed working with me. Engaging with each one of you has been an enriching experience.

Willis, you have been a constant source of support, and thank you for gifting me with your beautiful friendship. I thoroughly enjoyed spending time with you, especially our discussions and tasty eat-out meals. Soni, Rashmi, Rudra, and Sonali for being such good friends. Arittri for all the nice jamming sessions and for being a truly great friend.

In the end my family, mummy and papa for providing me with everything including the education that made me reach this far. My sister Nidhi jiji and Rahul jiju for being a constant source of infinite support and pushing me forward. Nikhil for just being there regardless of time or topic.

Babli for providing me with love & care, and being my anchor for the last six years. And Ishita, who played a crucial role in ensuring my mental well-being. Thank you all.

# List of publications used in this thesis

**Chapter 2 is based on the following publications**

1. Abhinav Singh, Pietro Incardona, & Ivo F. Sbalzarini (2021).
   A C++ expression system for partial differential equations enables generic simulations of biological hydrodynamics. The European Physical Journal E , 44, Article number: 117.

   The study concept was developed by AS under the supervision of PI and IFS. All the authors analyzed and interpreted the results. AS and PI implemented the software. All authors drafted the manuscript, and provided critical revisions.

   This publication is not currently used in any other dissertation, nor is it intended to be used in any future dissertations.

2. Abhinav Singh, Landfried Kraatz, Pietro Incardona, & Ivo F. Sbalzarini (2023).
   A Distributed Algebra System for Time Integration on Parallel Computers. Submitted to ACM Transactions on Mathematical Software.

   The study concept was developed by AS under the supervision of PI and IFS. AS, LK and PI implemented the software. All the authors analyzed and interpreted the results. All authors drafted the manuscript.

   This publication is not currently used in any other dissertation, nor is it intended to be used in any future dissertations.

**Chapter 3 is based on the following publications**

3. Abhinav Singh*, Philipp Suhrcke*, Pietro Incardona, & Ivo F. Sbalzarini (2022).
   A parallel numerical solver for active polar hydrodynamics and its application to active turbulence in three dimensions. Physics of Fluids 1 October 2023; 35 (10): 105155 (Cover Article).

   The study concept was developed by AS and PS under the supervision of IFS. AS developed the algorithm, implemented it in the software, performed benchmarks. PS implemented and performed additional benchmarks. PI helped with the software implementation. All authors interpreted the results. All authors drafted the manuscript.

   *Equal Contribution. This publication is not currently used in any other dissertation.

4. Abhinav Singh, Alejandra Foggia, Pietro Incardona, & Ivo F. Sbalzarini (2023).
   A Meshfree Collocation Scheme for Surface Differential Operators on Point Clouds. J

Sci Comput 96, 89.

The study concept was developed by AS under the supervision of IFS. AS developed the algorithm and implemented it in the software, and performed benchmark results. AF implemented and performed additional benchmarks. PI helped with the software implementation. All authors interpreted the results and drafted the manuscript.

This publication is not currently used in any other dissertation.

**Chapter 4 is based on the publication**

5. Abhinav Singh, Quentin Vagne, Frank Jülicher & Ivo F. Sbalzarini (2023). Spontaneous flow instabilities of active polar fluids in three dimensions. Phys. Rev. Research 5, L022061.

   The study concept was developed by AS under the supervision of FJ and IFS. AS developed the models, simulation methods. All authors analyzed and interpreted the results. All authors drafted the manuscript.

   This publication is not currently used in any other dissertation, nor is it intended to be used in any future dissertations.

**Chapter 5 includes experimental data from the following future publications**

6. Salman Alam, Bibi Najma, Abhinav Singh, Jeremy Laprade, Gauri Gajeshwar, Hannah Yevick, Aparna Baskaran, Peter Foster, Guillaume Duclos.
   Active Fréedericksz Transition in Three-Dimensional Active Nematic Droplets.
   Water in oil droplets of microtuble mixtures exhibiting spontaneous flows - Experimental data was obtained by Salman Alam at the Duclos Lab, Brandeis University, Massachusetts, USA.

7. Jianguo Zhao, Charlie Duclut, Abhinav Singh, Rahil Golipour, An Pham, Behzad Golshaei, Chonglin Guan, Mingru Li, Rudolf Oldenbourg, Stephan W. Grill, Ivo F. Sbalzarini, James L. Harden, Frank Jülicher & Christoph F. Schmidt.
   Active Force Driven Large-Scale Directional Flow of Isotropic Cytoskeletal Networks.
   The 1D spherically symmetric model simulations were done jointly with Charlie Duclut, MPI-PKS, Dresden, Germany.

# Contents

# Chapter 1

# Introduction

Biophysical phenomena's inherent complexity has been long subject to scientific scrutiny, with mathematical modeling serving as an instrumental tool in demystifying these systems [1–3]. Through meticulous and detailed mathematical models, we can simulate living systems around us to an astounding degree of accuracy, forecasting outcomes and enabling a profound understanding of nature [4, 5]. Notably, minimal models are sought after for their simplicity and ease of analysis [6], but they often fall short in accurately representing and predicting real-world conditions [7]. Prominent examples include mechanics of cells and tissues [8], where active hydrodynamic models have exhibited considerable success in recapitulating the spatiotemporal behavior of cellular migration to an astounding degree of accuracy. However, minimal phenomenological hydrodynamic models may fail to capture the instabilities in active microtubule mixtures confined to small channels [9, 10]. Another emerging class of models to study the mechanics of biological tissues is the discrete vertex models [11, 12]. Vertex models are discrete systems that evolve in time through free energy minimization and can capture chiral flows in rotating spherical organoids [13].

The use of Partial Differential Equations (PDEs) stands as a beacon in this complex terrain, offering substantial predictive power and the ability to model intricate phenomena via mean field variables [14, 15]. Notably, the Navier-Stokes equations, which delineate the equilibrium dynamics of fluid flow [16] serves as a foundation for continuum biophysical models such as the active hydrodynamic models. This class of models is particularly interesting due to its universal nature in explaining phenomena at scale from cytoskeletal dynamics, to tissue flows, and flocking behavior of animals. However, the utilization of such active matter models is far from straightforward [17, 18]. Nestled with the complexities of parameter fitting, mathematical and computational challanges, the study of different classes of models becomes hard. The intricate nature of PDEs calls for specialized numerical methods and optimized computational codes, resulting in a computationally demanding task, particularly in three-dimensional realms. Even though numerous solutions exist for studying a specific PDE, there remain various hurdles that impede progress in studying

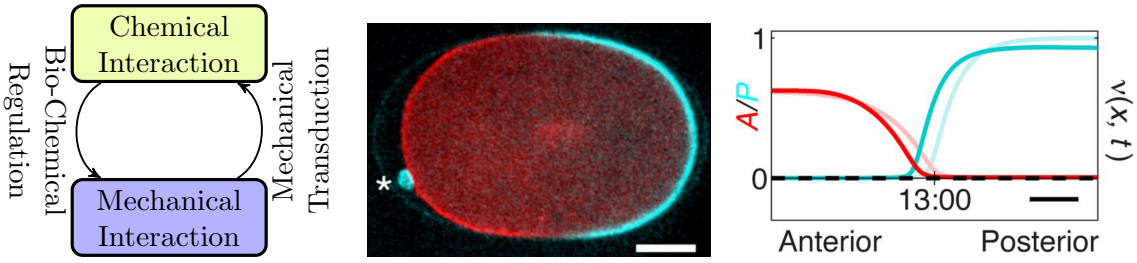
emerging PDEs.

The exploration of the new class of out-of-equilibrium active matter exemplifies such challenges [9, 19, 20]. The study of these materials involves dynamic systems like living matter across scales, whose model equations are still under investigation [21–23]. Such evolving and emerging PDEs necessitate the development of fresh, scalable codes, which becomes a bottleneck in the research process. The bottlenecks range from formulating a feasible numerical scheme that can accurately solve the model to implementing it in a high-performance computational environment. This work aims to propose innovative solutions to these challenges, providing a new perspective on modeling and simulation issues associated with a broad class of PDEs. Through a focused lens on out-of-equilibrium materials, we seek to shed light on the unknowns, pushing the boundaries of our current understanding and potentially unlocking new avenues for scientific progress.

## 1.1 Spatiotemporal modeling and simulation

The computational modeling landscape of physical systems is a vast and diverse arena, presenting models of varying complexity. It spans across the classical domain to the more advanced quantum realm, incorporating a rich variety of mathematical techniques and principles to accurately represent the behavior of matter. A key area of interest within this landscape is the modeling of living systems that are comprised of spontaneously flowing particles that interact with each other and their surroundings. Different modeling frameworks have emerged to provide a comprehensive understanding of this complex system, each with their unique insights and limitations.

Among the prominent modeling frameworks is kinetic theory [24], which offers a statistical mechanical representation of the behavior of these particles. Kinetic theories work by representing the dynamics of a large number of particles through statistical distributions, giving an aggregate picture of the system's evolution. This framework is particularly useful for understanding the collective behavior of active matter, where emergent phenomena arise from individual particle interactions. Active Brownian particles, on the other hand, provide a stochastic approach to model active matter [25]. They are characterized by a combination of directed motion and random fluctuations, where the strength of the fluctuations can lead to varied behavior, from ballistic to diffusive regimes. This model captures the interplay between the persistence of motion, the particles' self-propulsion, and the randomizing effect of the noise, which can lead to various emergent phenomena like clustering or phase separation. Symmetry-based theories [22], based on the Onsager principle [26], play a crucial role in modeling active matter as well. The Onsager principle of regression of fluctuations and reciprocal relations are built on the symmetries of the underlying system and offers a thermodynamically consistent bridge between microscopic dynamics and macroscopic behavior. This can be applied to active matter, where the non-equilibrium nature of the particles breaks detailed balance and leads to a rich array of




(a) (b) (c)

Figure 1.1: Spatio-temporal regulation in Biological Systems. **(a)** Biomehcanical feedback in biological systems. **(b)** A-PAR (red) and P-PAR (blue) proteins in *C. Elegans* embryo. **(c)** Concentration profile of Par proteins (opaque), and observed flow (translucent) in (b) on a one dimensional surface representing the embryo membrane. (b-c) are adapted from [27].

potential symmetries to explore.

These diverse modeling frameworks converge on a common understanding of active matter, highlighting the importance of the continuum assumption. This assumption simplifies the description of these systems by treating them as a continuous medium instead of discrete particles. While this may not always precisely reflect the microscopic behavior, it provides a powerful tool for understanding and predicting the large scale hydrodynamics of active matter, while also allowing for manageable computational complexity. Therefore, the blend of different modeling strategies, coupled with the continuum assumption, provides a more comprehensive and holistic approach to the study of active matter and its fascinating complexities.

One example of such mathematical models is the uni-dimensional advection-diffusion-reaction equation that can describe the dynamics of two chemical species A-PAR and P-PAR proteins, responsible for governing the establishment of cell polarity during development of a *C. Elegans* embryo [27]. The PAR proteins flow and diffuse on a curved membrane and interact with the bulk of the embryo via chemical reactions as shown in Fig. 1.1a-c. It is remarkable that the dynamics of the proteins can be recovered by a simple one dimensional advection-diffusion-reaction PDE,

$$\frac{\partial A}{\partial t} = \mathbf{v} \cdot \nabla A + \nabla^2 A + R_A(A, P) \tag{1.1a}$$

$$\frac{\partial P}{\partial t} = \mathbf{v} \cdot \nabla P + \nabla^2 P + R_P(A, P) \tag{1.1b}$$

describing the dynamics of proteins concentrations $A, P$ with reaction terms $R_A(A, P), R_P(A, P)$ respectively, that are advected with a known flow velocity $\mathbf{v}$ on a periodic line segment mapping the 2D surface. In reality, the embryo is three dimensional and undergoes more complex shape changes after polarity establishment. For

such simpler, lower-dimensional models, standard programming tools typically suffice to investigate and understand the protein concentration dynamics but the long time dynamics of the embryo is beyond the scope of this model. Adding complexity, the hydrodynamics can be modeled with an incompressible force balance in the system which takes form of an implicit Poisson equation with pressure $\Pi$ and additional force terms $f$,

$$\nabla^2 \mathbf{v} + \Delta \Pi = f \tag{1.2a}$$

$$\nabla \cdot \mathbf{v} = 0 \tag{1.2b}$$

that has to be solved with appropriate boundary conditions.

However, as we make the models more general to explain shape changes, and add the dynamics of the cytoskeleton, we ascend into higher dimensions. This leads to more intricate and demanding models, necessitating the use of specialized computational tools and numerical methods. To illustrate, consider the MinE/MinD protein oscillatory system found in the bacteria *E.coli* [28, 29]. State-of-the-art Finite Element Analysis (FEA/FEM) has shown potential to explain the three dimensional mechano-chemical interactions in this system [30, 31]. Often, a tailored numerical technique is selected and deployed within a scalable, black box computer code to tackle the problem. But such codes generally come with their limitations, being constrained to a specific PDE or model system and fixed numerical method. It is rarely the case that a computer code is implemented with the possibility of using the same code with a different numerical technique or domain geometry while maintaining scalability. Keeping the code efficient and scalable further burdens the computational study.

Despite this interconnections, it's important to recognize that model equations and numerical schemes are distinct concepts as shown in Fig. 1.2. They address different aspects of the computational modeling process, and the efficient, high-performance implementation of these schemes is another separate concern. Recognizing this, we leverage this separation to architect a system that is both modular and efficient, effectively disentangling these intertwined aspects. In the subsequent section, we will delve into specific model systems where the aforementioned challenges are palpable, showcasing the prowess of our proposed framework.

To provide a concrete example and drive the motivation behind our approach, we will initiate our discussion with an introduction to the captivating physics of active matter. This serves as a fitting starting point, setting the stage for the intricate exploration of emerging bio-physical PDE models in complex geometries.

## 1.2 Physics of active living matter

The origin of life in multicellular organisms is a fascinating enigma, a procedure as labyrinthine as it is mesmerizing [37]. Beginning from a single cell, the organism proliferates and orchestrates itself into intricate structures, demonstrating a spectrum of patterns.

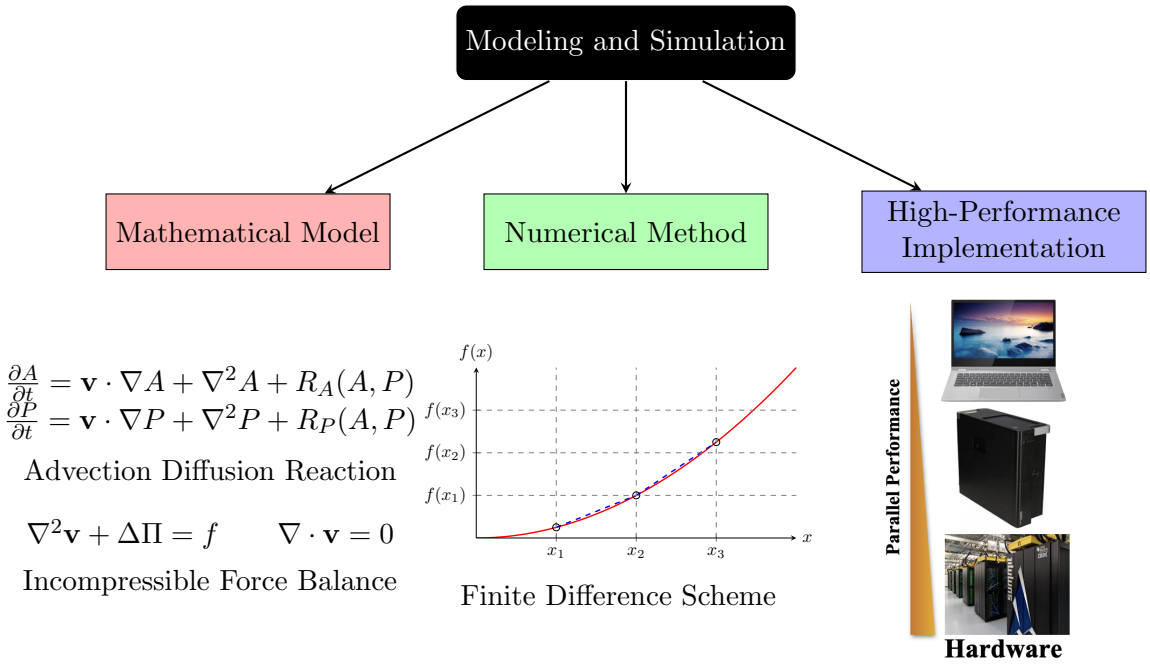



Figure 1.2: Demystifying black box modeling and simulation with *seperation of concerns*. Model equations (left), Numerical method (middle), Heterogeneous computing architectures (right).

This occurrence, identified as morphogenesis, is fundamental to grasping how life forms evolve. Notwithstanding countless experimental and theoretical investigations, a thorough comprehension of morphogenesis remains impenetrable. Pivotal questions endure: How do cells position themselves to construct complex structures? How can we formulate or simulate morphogenesis in 3D structures to anticipate an embryo's development? Deciphering these enigmas could offer a complete understanding of morphogenesis and the mechanical behavior of biological systems.

Model organisms and in-vitro model systems are instrumental in understanding biological systems through experimentation [38]. They offer direct comprehension into the dynamics of living materials that assist the mathematical modeling of the specific system. Moreover, biological systems display universal themes such as nematic ordering (molecules align in a common direction without specific positional order) and patterning across scales, as demonstrated in Fig. 1.3. Starting from a small scale of cytoskeletal organization, elements like microtubules and kinesin-1 motors showcase chaotic flows with nematic ordering and defect dynamics (Fig. 1.3a) [32]. Reconstituted actomyosin cortex generates active forces with measurable irreversibility (Fig. 1.3b) [33]. Cell cultures can arrange uniformly and

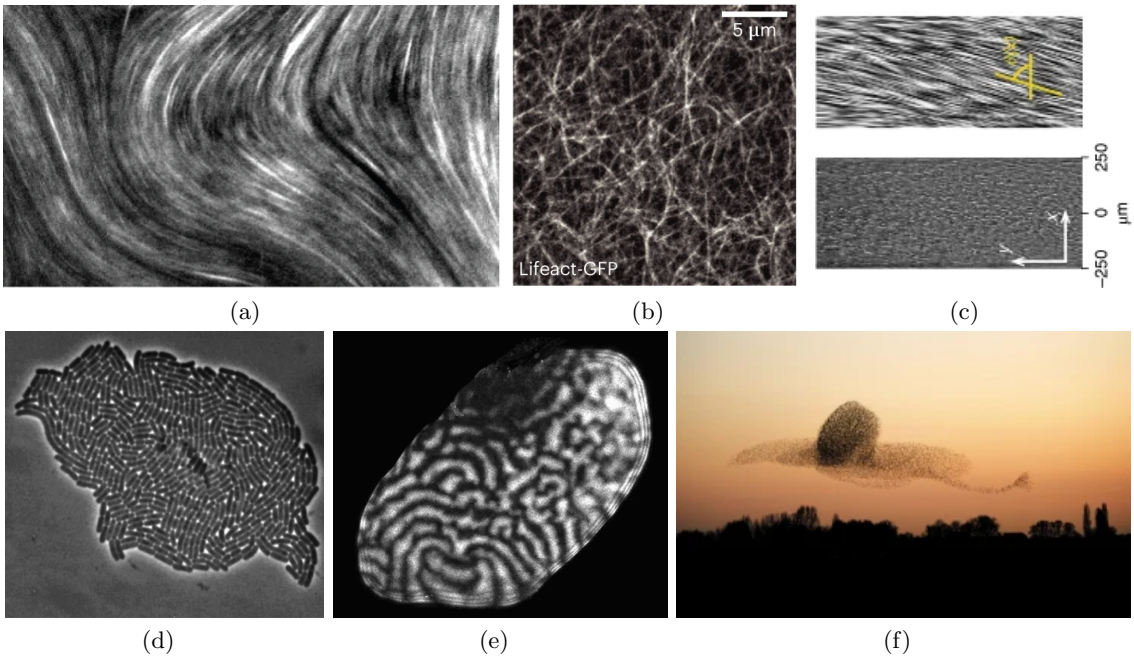



Figure 1.3: Order and patterning in out-of-equilibrium active matter across scales. **(a)** Mixture of microtubules and kinesin motors with turbulent nematic ordering [32]. **(b)** Reconstituted actomyosin cortex with quantifiable irreversibility [33]. **(c)** Spontaneous flow in spindle-shaped cell culture [19]. Uniformly aligned cells (bottom), spontaneously flowing cells (top). **(d)** A growing bacterial colony as an active nematic model system (Creative Commons CC BY ) [34]. **(e)** Spatio-temporal signaling patterns on the membarane of starfish embryo [35]. **(f)** Flock of birds with striking patterns [36]. All images are adapted from their respective reference with appropriate license permission.

break symmetry with spontaneous flow as an active analogy of the Fréedericksz transition (Fig. 1.3c) [19]. Bacterial colonies can organize and display coordinated flows (Fig. 1.3d) [34]. Starfish embryos show noticeable reaction-diffusion patterning with underlying order during development (Fig. 1.3d) [35]. At the multicellular organism scale, ants, fish, birds (Fig. 1.3e), and humans [39] show remarkable flocking order.

Thus, all biological systems share a common characteristic. They are inherently non-equilibrium systems that can be described with an order parameter, despite various behaviors. As Schrodinger famously noted, life avoids decay to equilibrium [40]. That's what makes these systems fascinating, having inherent behavior driven by the consumption of chemical energy including adenosine triphosphate (ATP). The principles of non-equilibrium physics are still nascent, and we are just beginning to decipher them. Notably, advancements have been made recently in observing living systems in-vivo [41]. For instance, scientists have

just started to quantify irreversibility in starfish embryos [35]. Out-of-equilibrium systems display measurable irreversibility and pattern formation. Cytoskeletal extracts also show similar irreversible dynamics [33].

While scalar order parameters such as a real number measuring degree of alignment in a system, are often investigated, vectorial order parameters are potentially overlooked due to the complexity of the modeling equations [42]. The cytoskeleton, where vectorial order emerges, provides an ideal setting to test active matter theories with scalar or vectorial order parameter. Notably, recent experiments have unveiled various instabilities in different cytoskeletal assays [9, 20, 43]. These assays consist of polymers and active molecular motors. The most frequently studied experimental model systems include microtubules with kinesin motor proteins, and the reconstituted actomyosin cortex where nematic order is ubiquitous.

### 1.2.1 Theoretical frameworks to study active fluids

Theoretical models that simplify the complex dynamics within a cell to predict shapes and pattern formations are of great value, given the enormous number of molecules within each cell. We now review a few popular models of active matter.

#### Brownian rods with active diffusivity (BRAD)

Brownian rods with active diffusivity (BRAD) models have emerged as a significant tool in understanding the complex dynamics of active matter systems [44–46]. At their core, these models focus on microscopic dynamics, describing the behavior of individual rod-like particles that display both self-propulsion and rotational diffusion - a form of random tumbling that can alter a particle's orientation. These systems can be exemplified by bacteria or synthetic swimmers [47], which navigate their environment in an active, self-driven manner while also being influenced by thermal fluctuations.

The interplay between self-propulsion and rotational diffusion gives rise to intriguing and diverse behaviors in these active systems. These individual dynamics, when combined with interactions among particles (such as steric or volume exclusion effects), ultimately shape the overall dynamics and behavior of the system. For instance, the balance between self-propulsion and rotational diffusion can influence the formation of clusters, the emergence of collective motion, or the transitions between different phases or flow states.

Although the BRAD model is not a hydrodynamic model per se [44], it serves a crucial role in bridging the gap between microscopic and macroscopic scales in active matter systems. The insights gained from BRAD models can be used to inform more extensive, continuum-level descriptions of the system. For example, one could derive a set of hydrodynamic equations that encapsulate the behavior of the system on a larger scale based on the understanding developed through the BRAD model. These hydrodynamic models capture collective dynamics and emergent phenomena such as large-scale flows or pattern formation, extending our understanding from the scale of individual particles to the behavior of the

system as a whole.

Thus, while BRAD models operate on a microscopic level, they lay the foundation for the development of macroscopic descriptions, contributing significantly to our comprehensive understanding of active matter dynamics. They demonstrate the critical link between individual particle behaviors and collective, system-wide phenomena, providing a robust approach to studying and interpreting active matter systems.

### Kinetic theories for modeling cytoskeletal dynamics

Kinetic theories have served as a valuable tool in modeling the dynamic behavior of microtubules [48, 49], which are a crucial component of the cell's cytoskeleton, known for their key role in maintaining the structure of the cell and facilitating intracellular transport. In this modeling framework, microtubules are represented as a distribution of rods that interact with each other and their surroundings, leading to complex, non-equilibrium dynamics. The kinetic theory provides a statistical description of these rod-like particles, defining a probability distribution function (PDF) for microtubules based on their location, orientation, and other factors, such as their length, which is especially important given the dynamic instability of microtubules. The dynamics of this PDF are governed by a kinetic equation, which contains an integral term representing the effect of collisions or steric interactions between microtubules.

The integral term is typically a multi-dimensional function of the microtubule distribution, leading to a hierarchy of equations that are challenging to solve. To overcome this issue, 'closures' are used [50], which are approximate solutions that truncate this hierarchy and close the system of equations, allowing for a more manageable computation [51]. Kinetic theories have the important advantage of proceeding from a microscopic model and therefore typically have coefficients that can be explicitly related to the microscopic model, such as the emergent shape and stresses in the material and viscosity of the surrounding medium. Kinetic theories can thus identify unexpected relations between physical coefficients [52].

Various closure approximations have been proposed for microtubule systems, each with their own advantages and limitations. For instance, the mean-field closure neglects correlations between microtubules and simplifies the system by assuming that each microtubule behaves independently of the others [53]. Although this simplification can lead to accurate predictions in some cases, it can miss important collective behaviors in densely packed systems. Pair-distribution closures [54] take into account pair-wise correlations between microtubules, capturing more detailed behaviors such as alignment due to excluded volume interactions or the formation of bundles. These models often introduce additional parameters, like pair distribution functions, that account for the orientation and distance between microtubules. For instance, an isotropic pair-distribution function might be used to model a system where microtubule orientations are uniformly distributed, while an anisotropic function could model a system with orientational order.

There is another layer of modeling that focuses on capturing the broader behavior of the system. Transitioning from the microscopic perspective, we turn our attention to symmetry-preserving hydrodynamical models. These models emphasize the conservation laws and symmetries inherent in the system, offering a macroscopic understanding of the active matter's behavior.

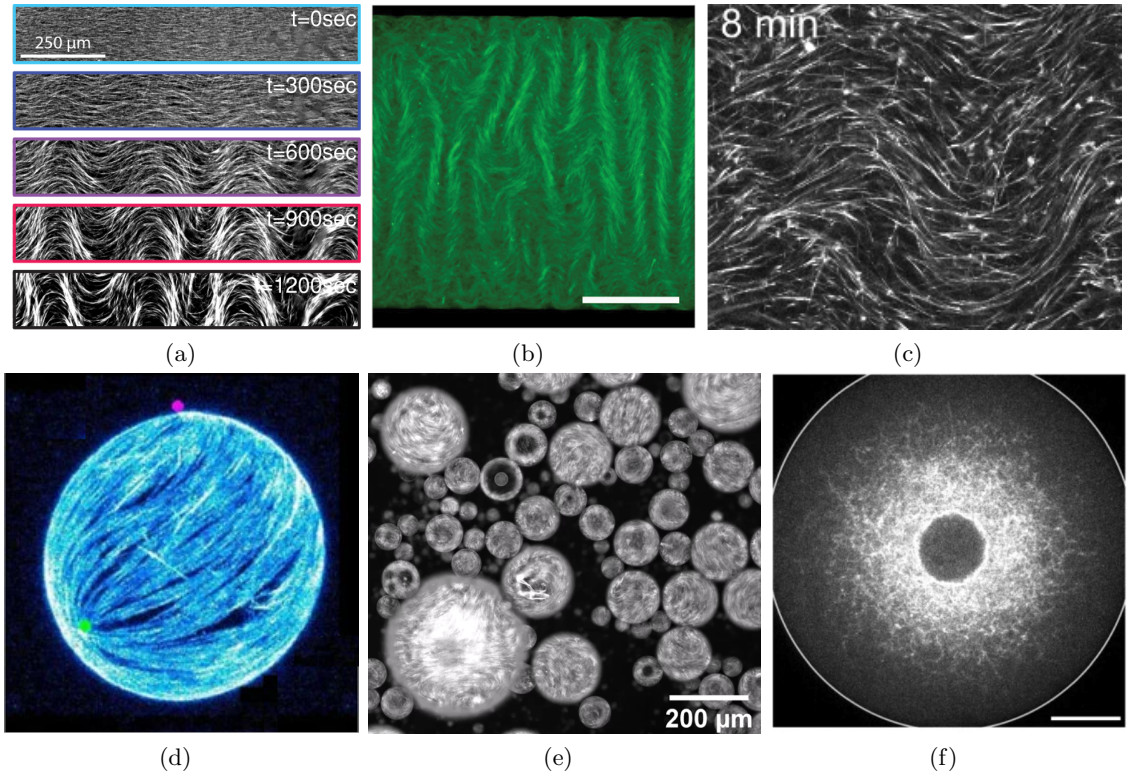
**A symmetry-preserving approach to cytoskeletal hydrodynamics**

A particularly successful model comes from the theory of active polar viscous fluids [21–23, 59–61]. This model can be derived from conservation laws that obey symmetries of the system. It is defined by a set of analytically unsolvable partial differential equations to represent a non-equilibrium system, embodying the movement of intracellular fluids along with the treadmilling process, and molecular motors generating active stress. It describes the dynamics of polar polymers (eg. microtubules/actin), coupled with molecular motors (eg. kinesin-1/myosin) that generate *extensile* or *contractile* stresses which ultimately generate forces that dictate the shape formation. Hydrodynamics models have been successful in explaining pattern formation in minimal settings with extensile active stress [9, 19, 20, 55], such as the bend instability of microtubulues as shown in Fig. 1.4a-c or spontaneous flow of cells in Fig. 1.3c.

The macroscopic behavior is crucial in the context of active matter, especially as it often correlates more directly with experimental observations. Experimentation in active matter primarily captures system-wide properties and emergent behavior rather than specific microscopic dynamics Fig. 1.4d-f. Therefore, a focus on macroscopic behavior provides an accessible and effective lens through which we can understand, predict, and potentially control the system dynamics. Symmetry-preserving hydrodynamic models are an ideal choice for this focus, given that they encapsulate the system's inherent conservation laws and symmetries. These models, informed by the Onsager principle and other such symmetry-based theories, allow us to predict and understand macroscopic phenomena as emergent properties of microscopic dynamics, bridging the gap between theory and experimental findings. In light of this, we will focus our study on these symmetry-preserving models, providing us with a pragmatic and insightful approach to the study of active matter. We describe the active Ericksen-Leslie hydrodynamics model in Chapter 3. However, studying the full model equations in real embryo-like geometries present significant challenges, exacerbated by the intricacies of three-dimensional modeling, the need for specialized numerical schemes, and the demands of high-performance implementation. We now outline such challenges.

### 1.2.2 Challenges associated with studying active hydrodynamics

Active hydrodynamic models, despite their great potential to shed light on complex non-equilibrium biological systems, still encounters a number of considerable challenges. The




(a) (b) (c)

(d) (e) (f)

Figure 1.4: Experimentally observed instabilities in confined cytoskeletal active matter **(a)** Mixture of microtubules and kinesin motors exhibit bend instability when aligned homogeneously [9]. **(b)** Similar mixture as in (a) but undergoes an out of plane winkling instability (Creative Commons CC-BY) [55]. **(c)** In and out of plane instability observed in crosslinked mixture of microtubules and motors [20]. **(d)** Dynamic structure of active nematic shells with active microtubules (Creative Commons CC BY) [56]. **(e)** Droplet size dependent spontaneous flow 3D active liquid crystals made with microtubules and rod like virus (Data provided by Salman Alam at Duclos lab) [57]. **(f)** Spontaneous flow in cytoskeletal extracts of *Xenopus* egg [58]. All images are adapted from their respective reference with appropriate license permission.

following will highlight some of the primary hurdles:

### Complexity of biological systems

The first challenge is simply the inherent complexity of biological systems. In real biological settings, cellular and sub-cellular processes may occur on complex curved surfaces [62, 63], and consist of many components interacting on multiple scales, from molecular to cellular

and multicellular levels. Each of these levels is incredibly complex, with many variables that need to be accurately accounted for. This multifaceted nature makes it difficult to construct an exact and comprehensive model.

**Computational load**

The mathematical models used in active hydrodynamics often involve solving a series of non-linear partial differential equations. These equations can be computationally expensive to solve, especially when applied to larger, more complex geometries in higher dimensions [64]. The challenge here is to develop more efficient algorithms and computational methods to make these calculations feasible in a reasonable time frame.

**Experimental verification**

A third challenge lies in the experimental verification of theoretical models [65]. Theoretical models must be grounded in experimental reality, yet the complexity and dynamism of biological systems often make it difficult to obtain precise and comprehensive experimental data. Moreover, there is no simple relationship between the active matter model parameters and the underlying molecular properties that directly relates to the experimentally observable quantities.

**Parameter estimation**

Accurate estimation of model parameters is another major challenge. In active hydrodynamics, the parameters often represent physical quantities that are difficult to measure directly, such as the coupling constants that scale forces generated by molecular motors or the viscosity of the cytoplasm. Therefore, determining accurate values for these parameters can be quite difficult.

**Vectorial and tensorial order parameters**

As earlier mentioned, while scalar order parameters such as the magnitude of the Q-tensors (alignment tensor of the nematic) are often investigated, full tensorial or vectorial order parameters are potentially overlooked due to the complexity of the modeling equations [23, 42]. While tensorial order parameters such as the Q-tensor based nematic models are previously studied, exploring vectorial order in the system could unravel new and unknown phenomena, but this requires both novel experimental methods and advancements in theoretical approaches to tackle the increased complexity in three dimensions [17].

Addressing these challenges will require a combination of improved experimental techniques, more sophisticated theoretical models, and advanced computational methods. As progress is made, the field of active hydrodynamics will become an increasingly powerful

tool for understanding the complex dynamics of living systems. In this thesis, we will particularly focus on addressing the computational and theoretical challenges.

## 1.3 Computational study of active hydrodynamics

The field of computational mathematics has provided a vast array of techniques to achieve *consistent* approximations of the numerical solutions to PDEs as shown in Fig. 1.5d.

Despite the emerging interest in utilizing machine learning methodologies for this task [66], conventional numerical methods continue to offer a more economical computational alternative. Consistency in numerical approximations is characterized by a steady decrease in approximation error correlated with an increase in resolution, thus indicating convergence. The issue of convergence in machine learning or deep learning models, however, is not yet fully understood and remains a topic of ongoing research [67].

Solutions to PDEs necessitate spatial and temporal discretization, which allows for the approximation of their evolution over the selected discretization. The concept of discretization, across all circumstances, provides a frame of reference for examining the continuous fields or variables. Traditionally, an Eulerian frame of reference is employed to observe variations within a continuous function via a fixed observational window. This concept, in essence, suggests that the location of discretization points remains constant over time. This particular attribute proves beneficial for numerical methods due to the preservation of the discretization structure. Nonetheless, the examination of a continuous field or variable that propels at a flow velocity $\mathbf{v}$ naturally suggests the relocation of the discretization point leading to better stability of a numerical solver. This induces modifications in the underlying discretization, including alterations in grid or mesh structures leading to a Lagrangian approach. It is crucial to note that both Eulerian and Lagrangian approaches carry their respective benefits and drawbacks pertaining to different modes of discretization.

This section will delve into the methodologies employed in structured discretizations such as meshes with equidistant nodes and interconnected nodes, as well as the utilization of particles or point clouds as a discrete representation of the continuous function satisfying the PDE. We will then discuss the discretization of time and solving PDEs both explicitly, and implicitly in time and space.

### 1.3.1 Mesh-based spatial discretization

Harnessing the symmetry of equidistant nodal grids for discretization has given rise to a class of numerical methods broadly categorized as mesh or grid-based methods (see Fig. 1.5a). These techniques transfigure the continuity of a domain into a systematized grid-like structure, where nodes are uniformly interspersed. This methodology paves the way for the streamlined application of time-honored numerical techniques including finite difference, finite volume methods, and spectral methods. However, when dealing with

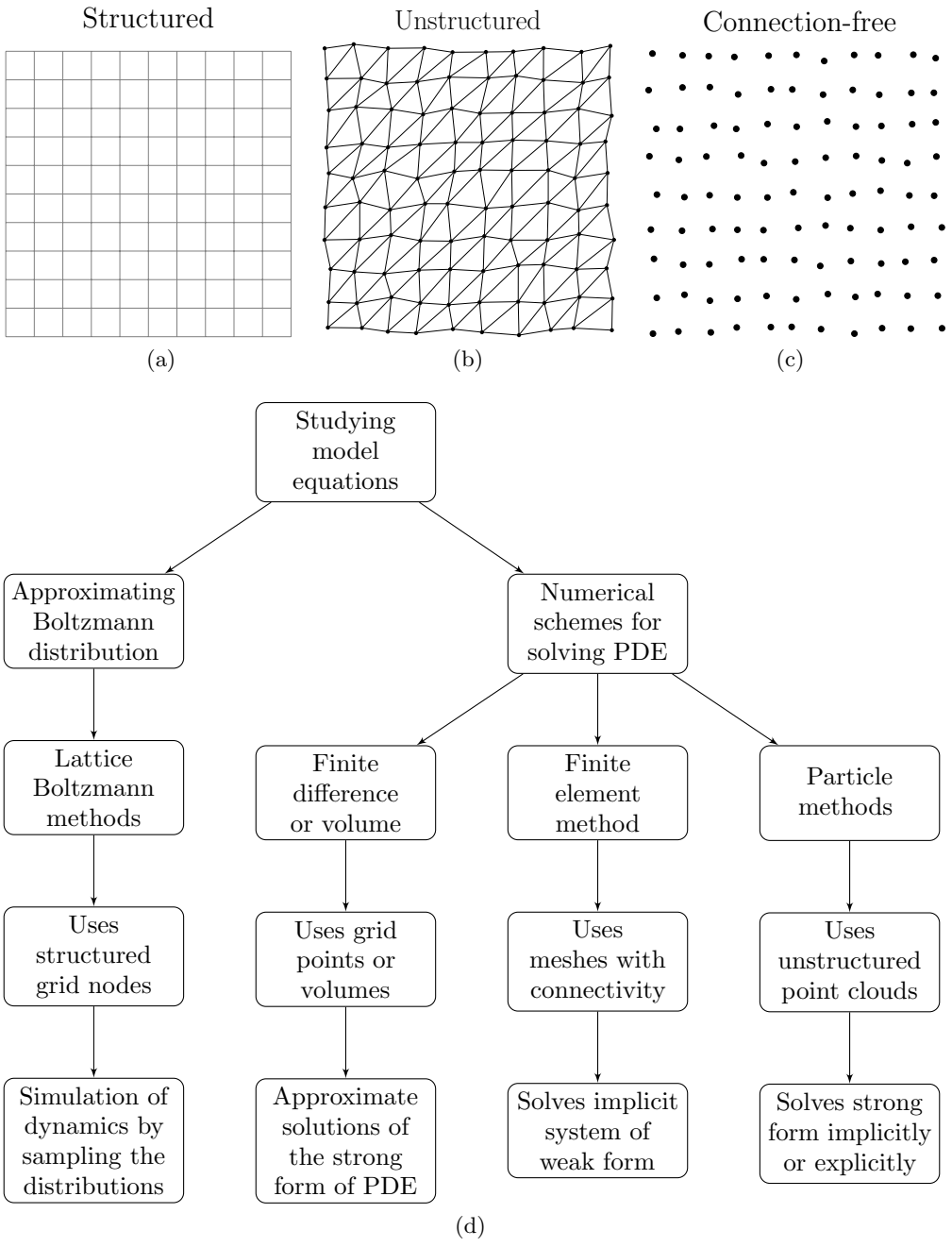



Figure 1.5: Underlying discretizations and computational methods for studying hydrodynamics numerically. **(a)** Structured Cartesian grid with equidistant nodes. **(b)** Unstructured mesh with connectivity. **(c)** Meshless particles with arbitrary positions. **(d)** Overview of numerical methods for approximating solutions of PDEs or simulating the dynamics of model equations.

domains that lack inherent symmetries or when the problem requires adaptive refinement, traditional equidistant grid methods are no longer sufficient. To tackle such situations, unstructured mesh or grid-based methods are employed. These methods are capable of handling complex geometric domains and offer the advantage of local adaptivity, making them a potent tool in numerically approximating solutions for a vast array of physical and mathematical problems. Consequently, they facilitate efficient computation for problems involving multi-scale phenomena, complicated geometries, or variable material properties.

Within the framework of structured meshes and extensions to unstructured meshes, the Finite Element Method (FEM) presents itself as a significant tool of considerable potency [68]. FEM, typically associated to irregular mesh with connectivity or mesh configurations (see Fig. 1.5b), maintains its utility across a spectrum of discretization scenarios. However, its application necessitates the transformation of complex PDEs into their weak forms or equivalent integral representation, which poses a significant challenge given the nonlinear couplings exhibited in emerging PDEs, as reflected in Eqs. (3.1), especially in the context of Lagrange multipliers. Following the formation of the weak PDE, the domain is compartmentalized into numerous minute, interconnected elements. This results in a substantial but sparse system of linear equations, solving which requires global communication across the domain due to its interconnected nature.

Despite these complexities, the deployment of FEM within a mesh discretization framework can prove beneficial, particularly when addressing problems that exhibit uniform geometric domains or where a consistent resolution is preferable [69]. FEM's capacity to boost precision, manage boundary conditions and enhance the accuracy of numerical solutions reaffirms its value. Nonetheless, mesh-based methodologies, inclusive of FEM, encounter hurdles when negotiating with intricate moving geometries or scenarios demanding adaptive remeshing due to the inherent rigidity of structured, fixed grids.

### 1.3.2 Particle-based spatial discretization

Offering a departure from grid-based methodologies, meshless or particle-based methods introduce an alternative means of modeling the continuous domain for solving PDEs. Particle methods employ a series of discrete points or particles scattered across the domain [70] without explicitly requiring the connectivity information (see Fig. 1.5c) to discretize differential operators. This flexibility allows for superior management of complex geometries and mobile boundaries. For example, a curved surface can be represented by a sparse set of points instead of a grid with a signed distance function. Particle methods typically build upon particle function approximations:

$$f(\mathbf{x}) = \int_{\Omega} f(\mathbf{y})\delta(\mathbf{x}-\mathbf{y})\mathrm{d}\mathbf{y} \tag{1.3}$$

by invoking a Dirac-delta function $\delta(.)$. For numerical application, a comparable kernel function $\eta_\epsilon(.)$ is employed:

$$f(\mathbf{x}) \approx f_\epsilon(\mathbf{x}) = \int_\Omega f(\mathbf{y})\eta_\epsilon(\mathbf{x}-\mathbf{y})\mathrm{d}\mathbf{y}. \tag{1.4}$$

Despite their flexibility, these methods present unique challenges, including the complexities of defining a local point neighborhood and the need for efficient algorithms to identify proximate points. The convergence properties of specific particle methods to discretize differential operators, such as Smoothed Particle Hydrodynamics (SPH) [71,72] and Particle Strength Exchange (PSE), are subject to scrutiny as the numerical error after discretization of Eq. (1.4) is limited by the quadrature error. Further, it is necessary to track a volume approximation and kernels do not converge with non-homogeneous neighbors such as single sided neighborhoods. Nevertheless, particle methods, including SPH, have made significant strides in a variety of areas, despite these obstacles [73–76].

A breakthrough in particle methodology has been achieved with the development of Discretization Corrected Particle Strength Exchange (DC-PSE) [77, 78]. This technique effectively addresses the convergence issues associated with traditional particle methods by generalizing finite differences to moving Lagrangian particles with consistency. Achieved by adjusting the kernel $\eta$ based on the location of neighboring points, DC-PSE guarantees convergence. Although the kernel adjustment necessitates additional computations within a neighborhood, the highly distributable nature and improved accuracy of these computations due to their localized scope makes DC-PSE an attractive option for consistently solving PDEs across arbitrary particle distributions and complex boundary conditions [79]. Hence one of the main thrust of this thesis will be to investigate the application of DC-PSE in simulating the active matter hydrodynamics within complex geometries.

### 1.3.3 Lattice Boltzmann method

The Lattice Boltzmann Method (LBM) is an approach that provides a complementary framework to the mesh-based and particle-based methods previously discussed. Unlike conventional fluid dynamic methods, LBM treats the fluid as a group of fictitious or pseudoparticles, allowing for the calculation of macroscopic quantities through statistical methods [80, 81]. The Boltzmann Transport Equation (BTE) forms the basis for the LBM. BTE is a statistical equation that describes the behavior of a thermodynamic system not in terms of macroscopic variables such as temperature and pressure, but instead in terms of the probability distribution of the positions and momenta of the gas particles. The key principle that it leverages is that the macroscopic fluid behavior arises from the collective, statistical behavior of large ensembles of individual particles.

In LBM, the computational domain is discretized into a regular lattice with interconnected nodes, similar to the structured mesh methods. However, unlike these methods, LBM considers not just the scalar or vector fields at these nodes, but also the

distribution function, a measure of the probability of pseudoparticles traveling in a certain direction. Time evolution in LBM is managed through two fundamental steps: collision and streaming. In the collision step, pseudoparticles at the same lattice node exchange momentum, simulating the effect of fluid viscosity. In the streaming step, pseudoparticles move from one node to another, thus conveying the convective flow of the fluid. One of the defining features of LBM is its ability to naturally incorporate complex boundary conditions and fluid interfaces, making it highly adaptable to problems involving complex geometries or multiphase flows. LBM is inherently parallelizable, providing a significant advantage in large-scale and high-performance computations. However, LBM also has limitations and is often seen as less accurate than traditional methods for some applications, primarily due to the limited velocity space discretization and numerical instabilities at high Reynolds numbers.

At the mesoscopic scale, the use of hybrid Lattice Boltzmann methods (LBM) serves as an attractive approach for exploring issues related to active matter, thanks to their straightforwardness [82]. These techniques stem from an algorithm originally designed for passive liquid crystals [83–85], which later expanded to cater to active matter and address a variety of problems such as chaotic flows, and defect dynamics within activity gradients [86–88]. However, the application of these methods has been confined to either periodic or no-slip flow boundary conditions, and the reproduction of Navier-Stokes equations is assured only for specific lattice symmetries [89] and distinct equilibrium distribution moment constraints [82, 86]. Furthermore, lattice Boltzmann simulations only achieve first-order convergence for curved boundaries [90] and currently, an open-source implementation along with a convergence study for 3D active hydrodynamics in the literature is missing. In addition, the hybrid character of these methods indicates that they depend on the time integration schemes employed for the evolution of the nematic or polar order parameter. We will now delve into the two primary types of time integration schemes.

### 1.3.4 Explicit time integration techniques

Having discretized the spatial operators, the remaining task to solve spatiotemporal partial differential equations (PDEs) involves discretizing time. This is a crucial step as it transforms a continuous problem into a discrete one, suitable for numerical approximation. A commonly employed technique for time discretization is the Euler method, characterized by its simplicity and ease of implementation [91].

In the Euler method, the solution at a certain time $t$ is used to approximate the solution at a subsequent time $t + dt$, where $dt$ denotes the time step size. This is achieved by utilizing a numerical time derivative at time $t$. This particular approach exemplifies an explicit method in that the future is inferred exclusively from knowledge of the past. To be more specific, a known solution (or a close approximation thereof) is employed to advance the system in time.

For instance, Eq. (1.1) can be discretized with the Euler method with $A_t$ as the numerical

approximation of $A(t)$ :

$$A_{t+dt} = A_t + dt\big(\mathbf{v} \cdot \nabla A_t + \nabla^2 A_t + R_A(A_t, P_t)\big) \tag{1.5a}$$
$$P_{t+dt} = A_t + dt\big(\mathbf{v} \cdot \nabla P_t + \nabla^2 P_t + R_P(A_t, P_t)\big). \tag{1.5b}$$

A variety of explicit techniques exist for time integration. However, these are frequently prone to numerical stability issues, where the solution diverges for certain oscillatory dynamics. Further, non-linear advection operator also exhibits instabilities when centered differences are coupled to Euler time steppers. Potential solution involve using upwinded operators, and higher order multistep/multistage time steppers, which combined with predictor-corrector methods, can mitigate such stability issues. Even then the time step is limited by the Courant-Friedrichs-Lewy (CFL) condition, which sets the upper limit for the time step size to ensure the solution's stability.

Nonetheless, testing different techniques swiftly in high-dimensional simulations presents a significant challenge, requiring computationally scalable code that implement stable time integration schemes.

### 1.3.5 Numerical solution using implicit stepping

Implicit methods comprise another category of time discretization techniques, widely recognized for their superior stability properties [92]. The fundamental concept of these methods involves discretizing time using an unknown future approximation, leading to a series of constraints. The solutions to these constraints are then sought, transforming the original PDE into a system of algebraic equations.

However, this seemingly straightforward process becomes non-trivial when dealing with non-linear terms, which remains an active field of research. Non-linear solvers may face convergence issues and are often optimized for specific PDEs.

For instance, the advection-reaction-diffusion Eq. (1.1) might contain non-linear reaction terms. Constructing an algebraic solver based on:

$$A_{t+dt} = A_t + dt\big(\mathbf{v} \cdot \nabla A_{t+dt} + \nabla^2 A_{t+dt} + R_A(A_{t+dt}, P_{t+dt})\big) \tag{1.6a}$$
$$P_{t+dt} = A_t + dt\big(\mathbf{v} \cdot \nabla P_{t+dt} + \nabla^2 P_{t+dt} + R_P(A_{t+dt}, P_{t+dt})\big) \tag{1.6b}$$

is a non-trivial task that may hamper the exploration of emerging models in computational physics. Thus, while implicit methods can offer superior numerical stability, their implementation is generally more complex and computationally demanding, limiting their application in large-scale, high-dimensional problems.

### 1.3.6 Application of implicit solvers for steady state PDEs

While the complexity and computational demands of implicit methods can be challenging, there are situations where their use becomes necessary. Specifically, for certain steady state

PDEs, such as the force balance Eq. (1.7), the use of an implicit solver is warranted [17].

$$\nabla^2 \mathbf{v} + \Delta \Pi = f \tag{1.7a}$$

$$\nabla \cdot \mathbf{v} = 0. \tag{1.7b}$$

The steady state nature of such equations implies that the quantities of interest such as velocity $\mathbf{v}$, and pressure $\mathbf{\Pi}$ do not change over time. Consequently, the time derivative vanishes and the equation simplifies to a spatial problem only. For these types of equations, an implicit solver has the distinct advantage of avoiding time step restrictions that are typically associated with explicit methods.

In the context of the force balance equation, an implicit solver constructs a system of linear equations, a direct consequence of discretizing the differential operator. This, in combination with the appropriate boundary conditions, allows for a robust numerical solution.

Thus, despite the computational challenges associated with implicit methods, their inherent stability and suitability for certain types of PDEs make them an indispensable tool in the numerical solution of certain stiff physical systems.

## 1.4 High-Performance computing in 3D modeling

To numerically solve PDEs, high-performance computing (HPC) plays a pivotal role, particularly when it comes to the complexity and sheer computational demand associated with three-dimensional (3D) modeling [93]. This domain involves the numerical approximation of partial differential equations (PDEs) with techniques as described in the previous section to simulate real-world phenomena, which often exhibit nonlinear characteristics in complex geometries.

High-performance distributed computing presents the ability to carry out a large number of computations rapidly, which is indispensable in the realm of 3D modeling. This is particularly relevant when dealing with high-dimensional and nonlinear PDEs, as they are inherently computationally expensive and can demand significant memory resources due to their increased complexity and the necessity of discretization over a voluminous 3D domain.

Over time, several numerical libraries have been developed to facilitate the utilization of HPC in solving PDEs with distributed computing parallelism using the Message Passing Interface (MPI) [94]. This includes frontend libraries such as OpenFOAM [95], Fenics [96], aLENS [54] and Dedalus [97], with backend libraries focussed on solving linear systems resulting from discretization including PETSc [98], Trillinos [99], UMFPACK [100] and Mumps [101]. The frontend libraries provide specific numerical methods and algorithms to aid in solving large-scale, complex numerical problems. The principal aim is to provide scalable solutions, that is, solutions where the computational time scales linearly with the

problem size and increasing computational resources, thus taking full advantage of the power of HPC.

However, employing these libraries efficiently is not a trivial task. It requires an in-depth understanding of the computational methods, the library architecture, and the specific problem to be solved. Additionally, these libraries tend to be geared towards certain numerical methods. While this ensures optimal performance for those methods for certain popular PDE models, it limits the flexibility when attempting to solve emerging PDEs that do not fit well within the predefined structures or when the application demands a different numerical approach. Emerging PDEs such as the active hydrodynamics often exhibit nonlinear couplings (see Appendix B), variable coefficients, and complex boundary conditions that make it difficult to develop efficient and accurate numerical methods using the existing tools.

One might consider developing their own customized computational tools to better handle these emerging PDEs. However, this approach brings its own set of challenges. Designing, developing, and validating a numerical software package is a time-consuming process that requires substantial expertise in both numerical methods and software engineering. Furthermore, ensuring that the software is efficient, accurate, and scalable to take full advantage of HPC resources adds another layer of complexity to this task. While high-performance computing is critical in 3D modeling of PDEs, writing scalable high-performance computer code is a complex task. As shown in Fig. 1.2, leveraging the separation of concerns abstraction can lead to the development of more flexible, user-friendly, and efficient. It suggests that HPC tools can be separated from the numerical methods and the model equations. One such attempt is the OpenFPM library for scalable scientific computing [102], greatly enhancing our ability to tackle the numerical challenges posed by the 3D modeling of emerging PDEs.

### 1.4.1 OpenFPM - open framework for particle and mesh codes

OpenFPM, the open framework for particle and mesh codes on heterogeneous architectures, is a C++ library that aids in the creation of distributed simulation codes [102]. The primary objective in designing OpenFPM was to maintain scalable performance without compromising portability, a task that is intrinsically challenging given that performance optimization typically reduces portability, and vice versa [103].

Uniquely, OpenFPM employs cutting-edge template meta-programming techniques to create distributed data structures that are performance-portable across heterogeneous architectures [64, 104], and allow for low-level memory access into the data structure transparently. This is a distinguishing feature when compared with other scalable computing libraries, as they often do not allow modifications to lower-level primitive data structures, like the arrays underlying discretization structures such as meshes. This constraint, while maintaining scalability, can restrict the customization needed to solve specific partial differential equations (PDEs) that call for such granular modifications.

Additionally, OpenFPM's design is characterized by high modularity and object orientation, simplifying the process of adding features and creating scalable abstractions. This design principle facilitates the implementation of the methods depicted in Fig. 1.5d with ease. This is primarily due to the submodule `openfpm_data` [104, 105], that allows for transparent memory and compute abstraction across CPU and GPU architectures. Given the open-source nature and modular structure of the library, implementing new numerical methods in `openfpm_numerics` and maintaining readability of the code, all while preserving scalability, becomes a straightforward task.

In this thesis, I will explore how we utilize OpenFPM to engineer abstractions that ensure separation of concerns of model equations, numerical methods, and scalable computer code, as illustrated in Fig. 1.2. This approach fosters the creation of modular and concise codes that can solve PDEs efficiently using techniques like the Finite Difference or Particle Methods, outlined in Fig. 1.5d.

## 1.5 Thesis blueprint

This thesis takes a deep dive into the multifaceted challenges we encounter while solving spatio-temporal Partial Differential Equations (PDEs) in complex geometries that arise when studying biophysical systems. While our primary focus rests on active living matter, we design our work to have universal applicability, paving the way for advancements in the scientific exploration of an expansive array of physical models.

**Chapter 2: C++ template meta programming for solving PDEs**

In Chapter 2, we harness the power of C++ template meta-programming for the analysis of complex PDEs. We adopt a strategic approach by separating concerns and spotlighting the design abstractions introduced in the OpenFPM library, fostering a transparent and scalable scientific computing environment. We create a template expression system that permits coding in a notation nearly akin to mathematical notation of the PDE. Additionally, we illuminate the applications and statistics when juxtaposing this novel approach with traditional codes, presenting a compelling case for modernization.

**Chapter 3: Numerical schemes for solving bulk active hydrodynamics and surface PDEs**

In Chapter 3, we unfold the complex world of active hydrodynamics and derive the generic hydrodynamics theory in three-dimensions. We innovate by proposing particle and hybrid particle-mesh algorithms to decode the hydrodynamics of active matter. We transparently share the hurdles we face and illustrate how the expression system, developed in Chapter 2, simplifies high-performance implementation. We affirm our findings through extensive

validation of the numerical code in 3D channels and validate our Stokes flow solver in a 3D Ball geometry.

We further present a new method named Surface DC-PSE, for investigating partial differential equations on curved surfaces. We design a meshless scheme to consistently discretize surface differential operators, like the Laplace-Beltrami operator, and apply it to compute mean and Gaussian curvatures. Our method stands on the foundation of an embedding approach, which can be analytically pared down to the implicit surface, provided the normal. We establish that our method can solve implicit equations on curved surfaces consistently, and present evidence of its convergence. We employ our expression system from Chapter 2, demonstrating its adaptability to fresh schemes.

**Chapter 4: Active hydrodynamics in 3D channels**

Using our model and simulations, we unravel the mystery of all cytoskeletal instabilities and their dependence on boundary conditions. We find a three-dimensional spontaneous flow transition, an active analog of the Fréedericksz transition using linear perturbation analysis. We present comprehensive phase diagrams of the spontaneous flow transition, numerically confirming its existence. We further find a topological phase transition, similar to the Berezinskii–Kosterlitz–Thouless (BKT) transition in the 2D XY model, that occurs after the spontaneous flow transition on increasing activity. We show how an active fluid transitions to turbulence and predict existence of traveling waves, periodic intermittency and spatio-temporal chaos in active fluids by numerically quantifying the maximum Lyapunov exponent.

**Chapter 5: Active hydrodynamics in 3D complex geometries**

Chapter 5 is a showcase of our numerical simulations of active hydrodynamics in intricate geometries, based on the algorithm we developed in Chapter 3. We start by showcasing simulations of active nematic balls in 3D. Here we again find the spontaneous flow transition but for much higher critical activity. We apply our insights to understand spontaneous flow in annular geometries. We broaden our horizons by exploring a density-based model for spontaneous flow in spherical shell-like geometries, thereby celebrating the versatility and flexibility of the expression system we innovated in Chapter 2.

**Chapter 6: Conclusion and future vision**

In our closing chapter, Chapter 6, we tie together our findings, affirming particle methods and OpenFPM as a scalable numerical framework for studying biophysical phenomena. We make a robust case for our novel approach, echoing our journey through this thesis, from harnessing C++ meta-programming to the unveiling of Surface DC-PSE.

# Chapter 2

# C++ template meta programming for solving PDEs

## 2.1 Motivation

Simulation software implementations are typically specific to a certain numerical method and a certain model, with discretized differential operators often hard-coded in explicit program statements. Changing the simulated model, as often required in biological physics when new molecular interactions become known, therefore usually requires rewriting much of the simulation program. This comes with multiple challenges in terms of code structure and maintainability, but also with respect to the computational speed and parallel scalability of the implementation. Due to the complexity of the models, simulations of active biological hydrodynamics can be time-consuming if not implemented in a properly optimized way on modern parallel computer architectures. Particularly in the fields of biological hydrodynamics and soft matter physics, there is thus a clear need for generic simulation software platforms that cleanly separate the PDE model to be simulated from the numerical methods used to do so, and that automate the generation of fast and efficiently scalable computer programs.

Here, we present such a generic simulation environment based on the OpenFPM parallel computing framework [102]. It is based on a novel C++ template expression system for PDEs. This expression system allows specifying the PDE model to be simulated in near-mathematical notation and, importantly, independently of the specific numerical method to be used. The numerical method is transparently selected via includes and template parameters, which allows rapidly changing or further developing the numerical method without having to rewrite the model specification. It also allows changing the model, e.g., including additional terms in the PDE or changing the geometry of the solution domain, without having to change the implementation of the numerical solvers (albeit one may have to switch to a different solver). Our framework therefore cleanly separates the specification

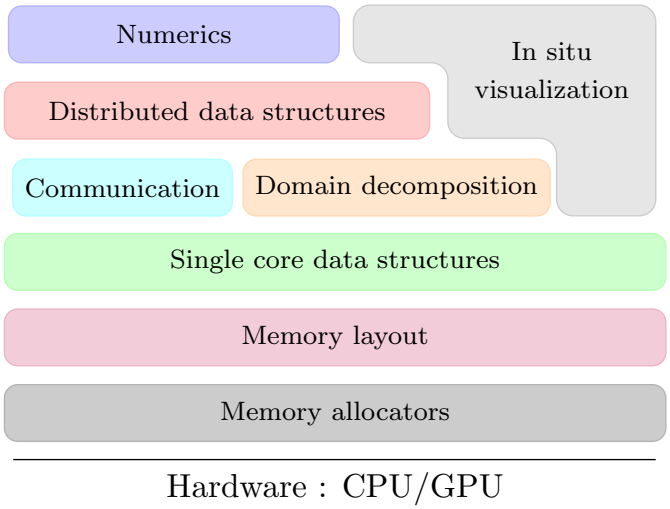



Figure 2.1: Layer structure of the OpenFPM stack.

of the continuous PDE model and of the numerical discretization method. This follows a famous principle from software engineering, *separation of concerns*, which is usually not followed in numerical simulation codes.

The idea of achieving separation of concerns using template expressions is not new and has already been implemented decades ago, e.g., in the Par-EXPDE project [106]. Hence, we design a system that is not be tied to a specific computer architecture and is specifically designed for continuum simulations of biological hydrodynamics. It therefore supports both particle-based and mesh-based PDE discretization, as well as hybrid particle-mesh methods [107]. Also, our template expressions transparently work for scalar, vector, and tensor fields, and they can be used to assemble equation systems for implicit solvers. Basing our system on OpenFPM also makes it portable across CPU and GPU architectures and provides numerical efficiency and parallel scalability.

After describing our framework, we illustrate how separation of concerns simplifies the implementation and maintenance of frequently occurring simulations in biological hydrodynamics and renders simulation codes less error-prone. In the application examples, we show how our framework allows changing the simulated model by altering just a few lines of code. We do so by changing the simulation from solving the incompressible Navier-Stokes equations, to simulating active polar fluids. We also show how our framework allows to change the numerical method by providing examples using both grid-based finite-difference methods and mesh-free particle methods.

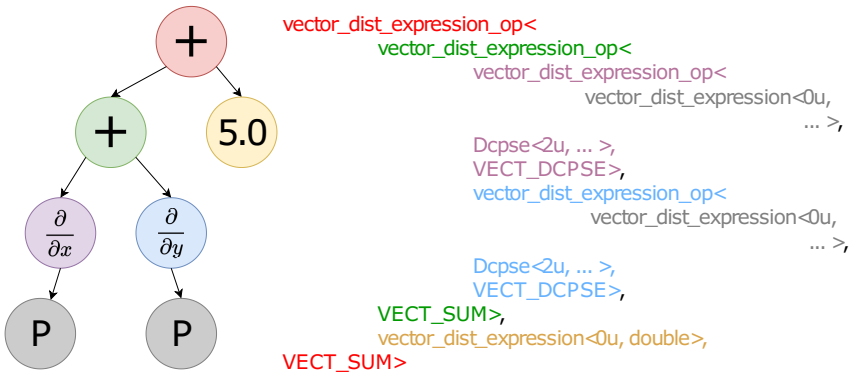



Figure 2.2: Example C++ template type representation of the expression $\frac{\partial P}{\partial x} + \frac{\partial P}{\partial y} + 5.0$.

## 2.2 Template expressions in OpenFPM

The C++ programming language allows for custom expression systems to be built using template expression parsing techniques [108]. We design and implement such a system for expressing PDEs in near-mathematical notation as C++ code and to specify the numerical methods to be used for their discretization using `include` statements and encapsulated interfaces to numerical solvers.

```
1 auto P=getV<Pressure>(particles);
2
3 Derivative_x Dx;
4 Derivarive_y Dy;
5
6 auto expression = Dx(P) + Dy(P) + 5.0;
```

Listing 2.1: Syntax example of the present expression system for the expression $\frac{\partial P}{\partial x} + \frac{\partial P}{\partial y} + 5.0$.

An example of the code this allows one to write is shown in Listing 2.1. It defines in line 1 a field `P` with automatically inferred datatype. The field `P` is linked to the values the property `Pressure` stored on a set of discretization points called particles using the function `getV`. In lines 3 and 4, two derivative operators are declared, `Dx` and `Dy` of type `Derivative_x` and `Derivative_y`, respectively. Model expressions can then simply be written as shown in line 6 for the example $\frac{\partial P}{\partial x} + \frac{\partial P}{\partial y} + 5.0$, providing a very human-readable notation that automatically extends over the potentially many points in the set `particles`.

This code is then analyzed and expanded to standard C++. Code analysis is done using the C++ expression system parser. In this parser, each expression is represented as a tree whose leaves (i.e., terminal nodes) are variables or numbers. Any interior (i.e.,

non-terminal) node of the tree represents one mathematical operator. As an example, the expression tree for line 6 in Listing 2.1 is shown in Fig. 2.2 on the left. This tree is subsequently translated to class nodes by C++ production rules, which we adjust for our needs by operator overloading. To do so, we define templated class nodes representing the basic binary operators $+$, $-$, $*$, $/$, and derivative operators. These are inserted at the location of the respective tree node to generate the typed code as shown in Fig. 2.2 on the right.

Terminal nodes represent C++ objects linked to a particular continuous field in the PDE, or to a numeric constant. They map to OpenFPM's abstract data types, such as distributed vectors (`vector_dist` in Fig. 2.2) or meshes.

Each non-terminal node of the expression tree has 3 template parameters. For binary operators, the first template parameter is the expression on the left-hand side of the operator, the second parameter is the right-hand side expression of the operator, and the third parameter indicates the type of operation (e.g., $+$, $-$, $*$, $/$). An operator can either be applied to the result of another operator, or to a variable or constant in a terminal node. We implement both binary operators and unary operators (e.g., derivatives). For the unary derivative operator, the second template parameter is the C++ class encapsulating the code used to discretize the derivatives, i.e., the numerical method (`Dcpse` in Fig. 2.2). For example, the production rule of the node $+$ in the expression $\frac{\partial P}{\partial x} + \frac{\partial P}{\partial y}$ is implemented by overloading the binary `+` operator for the two child nodes. In the example of Fig. 2.2, both child nodes are of type `vector_dist_expression_op`$< \ldots >$ (purple and blue in Fig. 2.2). The thus-overloaded `+` operator returns the type for the sum of the two derivatives (green in Fig. 2.2). For objects of class `Derivative`, we additionally overload the `()` operator. This enables production rules for expressions like `D(f)` or `D(f+g)`, denoting the derivative operator applied to the expression given inside the parenthesis as an argument to the unary `()` operator. This is how the purple and blue types in Fig. 2.2 are created, which in turn serve as input the to `+` operator.

In order to compute the result of the expression, each tree node defines a function `value`. Evaluating the function `value` of any node computes and returns the results at this node. This is done by recursively evaluating the operators on lower tree nodes as required by the natural tree traversal order. For computational efficiency, all nested calls to `value` functions along the tree are inlined. For the example from Listing 2.1, the data are the appropriately processed values of the field variable `P` at the locations of the discretization points in the set `particles`. Therefore, evaluating the function `value` of the root node of an expression tree produces the final result of the numerical evaluation of the given expression. For convenience, e.g., to simplify memory allocation or calls to external solver libraries, each node also provides a function `value_nz`, which returns the indices of the collocation points with non-zero data values that correspond to stencil-like coefficients required to build an implicit system of linear equations.

Using this C++ template expression system, complex equations involving multiple differential operators can be numerically evaluated in a transparent way. Using the above-

described programming techniques, we thus provide a content-aware expression parsing system based on computational graphs. This is then translated to distributed OpenFPM data structures, such as OpenFPM vectors and aggregates, to generate scalable distributed simulations. Implicitly, these internally distributed C++ objects are symbolic references to data, which enables expression-based computing.

```
1  // Define symbolic references to...
2  auto P=getV<0>(particles); // ...a scalar field
3  auto V=getV<1>(particles); // ...a vector field
4  auto Stress=getV<2>(particles); // ...a tensor field
5
6  // Assign values to components of fields for all particles
7  V[x]=1;
8  V[y]=0;
9  Stress[x][y]=5;
10
11 // Evaluate an expression and store the result
12 P = P + V[x]*V[x] + V[x] - Stress[x][y];
```

Listing 2.2: Expression-based computing with symbolic references.

An example of expression-based computing is shown in Listing 2.2. It defines symbolic references for a scalar field (line 2), a vector field (line 3), and a tensor field (line 4) from the properties of corresponding datatype stored on a set of discretization points called `particles`. Scalar values are then assigned to the components of the vector field (lines 7, 8) and to a component of the tensor field (line 9) across all discretization points. Components are selected by their symbolic name passed to the overloaded unary `[]` operator. Finally, an expression is evaluated resulting in a scalar field stored in `P` (line 12). Behind the scenes, this creates a computational graph and maps the computations required during compile time for efficient computation at run-time. The created expression class then enables expressions that are easily readable and writable even for complex PDEs.

### 2.2.1 Computing numerical differential operators

The numerical methods for discretizing continuous derivatives are specified by `include` statements. OpenFPM provides a library of frequently used numerical methods, including finite-difference methods on regular Cartesian grids and the DC-PSE method [77] for irregularly scattered discretization points. Users can implement additional numerical methods as plug-ins, which can then be used in the present template expression system as well. An example is shown in Listing 2.3, where the DC-PSE operator library is included in line 2. One can then instantiate correspondingly discretized versions of differential operators as shown in lines 6–10 for different examples. The first argument in the parentheses is the set of collocation points over which the operators are to be discretized. The second argument is a positive integer specifying the order of convergence for the discrete operator approximation. The third argument defines a cut-off radius for the operator support, i.e.,

the maximum range around a discretization point where neighboring points contribute to the discrete operator

```
// Import the discretization method from OpenFPM
#include "numerics/DCPSE_op.hpp"
// Listing 2 is included here
...
// Create discrete DC−PSE operators
Derivative_x Dx(particles,order,rCut);
Derivative_y Dy(particles,order,rCut);
Gradient Grad(particles,order,rCut);
Laplacian Lap(particles,order,rCut);
Advection Adv(particles,order,rCut);

// Compute Derivatives by applying these operators to symbolic expressions
V = Grad(P);
V[x] = V[x] + Dx(Stress[x][x]);
V[y] = V[y] + Dy(Stress[y][y]);
```

Listing 2.3: Including a specific numerical method and using correspondingly discretized differential operators.

We describe the expression template class that creates such differential operators in Appendix A.2. The expression system presented so far enables continuous models to be written in near-mathematical notation and to independently specify the numerical methods that shall be used to discretize them. This cleanly separates the model definition from the implementation of the numerical methods, as was the main motivation for our work. However, it so far only allows to evaluate explicit expressions, where the values can be computed successively along the expression tree. Further, in most spatiotemporal physical systems, expressions evaluations differs on the bulk and boundary of the system. This requirement motivates expression evaluation on only a subset of the domain. Hence, in the next subsection, we describe a subset framework to evaluate expressions on a particular prespecified partition of the domain.

### 2.2.2 Evaluating expressions on subsets

We develop a derived class for the distributed vector within OpenFPM to support partitioning of the particle set. These partitions can be viewed as unique subset discretization. This feature is particularly useful, as it is often required to compute expressions on a select subset of particles. An instance of this can be seen when applying time stepping only on the bulk when Dirichlet boundary conditions are set, effectively avoiding unnecessary computations. This class supports labeling the particles with a partition number. Next, a derived class of the distributed vector with partition numbers is used for construction of subset objects which correspond to various sub-discretizations such as the system's boundaries and bulk.

Note that in C++, an lvalue (locator value) refers to an object that occupies a identifiable location in memory, meaning it has an address that can be accessed and potentially modified, whereas an rvalue (read value) is typically a temporary object such as a literal or the result of an expression, which does not have a stable memory address and is primarily used as a value in assignments or calculations. Expressions formed using `getV` from the subset objects are permitted to be utilized as lvalues, but not as rvalues[1]. We showcase the subset expression system in the listing 2.4.

```
1  dist_subset_vector particles_bulk(0);
2  dist_subset_vector particles_boundary(1);
3  auto P=getV<Pressure>(particles);
4  auto P_bulk=getV<Pressure>(particles_bulk);
5  auto P_boundary=getV<Pressure>(particles_boundary);
6
7  Derivative_x Dx;
8  Derivarive_y Dy;
9
10 P_bulk= Dx(P) + Dy(P) + 5.0;
11 P_boundary=0;
```

Listing 2.4: Syntax example for evaluation of the expression $\frac{\partial P}{\partial x} + \frac{\partial P}{\partial y} + 5.0$ on a subset of bulk particles.

In listing 2.4, `particles_bulk` is a subset object created using a label '0' that was created during particle intiialization with the method `Particles.setSubset(0,p)` for particle with ID `p`. This enables the creation of a subset expression that evaluates only on a subset of the particles, as demonstrated in lines 10-11.

We take further steps to validate if subset expressions are only used as an lvalue. This verification is only conducted when the OpenFPM debug flag SE_CLASS1 is activated to avoid performance loss in production code. The subset class also provides vectors containing the ID of a subdiscretization, which we use for the assembly of implicit systems. We demonstrate this in the following subsection.

### 2.2.3 Distributed assembly of implicit linear systems with expressions

For implicit equations, direct evaluation is not possible. Instead, a numerical solver needs to be invoked to solve for the unknown values of the linear or nonlinear equation system. This is, e.g., the case when using implicit time-stepping methods in a simulation or when solving systems of linear equations in finite-difference simulations. Both involve first building the system matrix of a linear system of equations to be solved, and the right-hand side vector. These are then passed to a numerical solver, such as a multi-grid solver, an LU decomposition, or a Krylov subspace solver. Many such solvers are implemented in numerical

[1]This is a design decision to allow the formation of meaningful expressions and to avoid expressions with incorrect semantics. For instance, the addition of two subset expressions is nonsensical since the addition operation is not well defined for two different (sub)discretizations.

libraries like OpenFPM [102], PETSc [98], Eigen [109], and others, to which our framework provides transparent access through wrappers and a coherent interface.

An example is shown in Listing 2.5 to solve for the steady state of the Stokes equation. It includes a linear system solver based on DC-PSE in line 2. The solver is instantiated in line 9 for a simulation in two dimensions with two variables. The two equations for the two variables are created in lines 18–21, complete with their right-hand sides. This can be conveniently done using the symbolic expression system described so far. These equations are then imposed in the bulk of the simulation domain in lines 24 and 25. The boundary conditions are imposed in lines 27 and 28, here homogeneous Dirichlet zero-value boundaries. Finally, the numerical solver is executed in line 29, returning the solution.

```
1  // Import the solver from OpenFPM
2  #include "numerics/DCPSE_Solver.hpp"
3
4  // Listing 3 is included here
5  ...
6  // We assume indices of discretization points are stored in OpenFPM vectors 'boundary' for the
       boundaries and 'bulk' for the interior of the simulation domain.
7
8  // Initialize a system of 2 equations in 2D
9  DCPSE_scheme<equations2d2,particles> Solver(particles);
10
11 // Set unique variable and equation IDs
12 V.set_var_id(0);
13 eq_id vx, vy;
14 vx.setId(0);
15 vy.setId(1);
16
17 // Expressions for the two equations and their right−hand sides.
18 auto Stokes_x=Adv(V,V_star[x])+Lap(V[x])
19 RHS[x]=Dx(P);
20 auto Stokes_y=Adv(V,V_star[y])+Lap(V[y]);
21 RHS[y]=Dy(P);
22
23 // Impose the equations in the bulk
24 Solver.impose(Stokes_x,bulk,RHS[x],vx);
25 Solver.impose(Stokes_y,bulk,RHS[y],vy);
26 // Impose Dirichlet boundary conditions
27 Solver.impose(V_star[x],boundary,0,vx);
28 Solver.impose(V_star[y],boundary,0,vy);
29 Solver.solve(V_star[x],V_star[y]); //Solve
```

Listing 2.5: Using a numerical solver for an implicit equation.

To change this from a DC-PSE solver for irregularly scattered discretization points to a finite-difference solver on a regular Cartesian grid, only the following lines change:

```
1  // Changes in the initialization
2  #include "numerics/FD_Solver.hpp"
3  FD_scheme<equations2d2,grid> Solver(ghost,grid);
```

```
4
5  // Changes in the solver interface
6  Solver.impose(Stokes_x,start_id,stop_id,RHS[x],vx);
7  Solver.impose(Stokes_y,start_id,stop_id,RHS[y],vy);
8  Solver.impose(V_star[x],start_id,stop_id,0,vx);
9  Solver.impose(V_star[y],start_id,stop_id,0,vy);
10 Solver.solve(V_star[x],V_star[y]); //Solve
```

Listing 2.6: Changing to a mesh-based implicit solver using finite differences.

Instead of passing the OpenFPM distributed vectors `bulk` and `boundary` containing the respective subset, of discretization points, we now pass an OpenFPM mesh `grid`, where `start_id` and `stop_id` are the grid point indices of the bounding boxes of the respective regions. Alternatively, an OpenFPM mesh iterator could be used. All symbolic expressions for the model equations to be solved remain unchanged, despite the fact that we now use a fundamentally different numerical method, namely a Cartesian mesh finite-difference solver instead of a mesh-free DC-PSE particle method.

### 2.2.4 Temporary expression containers

Up to this point, we have used he OpenFPM distributed vector for evaluating and storing expressions. This approach isn't optimal, given that the OpenFPM distributed vector is templated, and thus the number of properties is fixed at the declaration of the distributed vector. Numerous scenarios necessitate temporary computations that aren't required to be stored in the distributed vector and don't necessitate ghost communication. For this reason, we introduce a new class, termed `t_exp<ST>`, which enables the creation of temporary placeholder expressions of type `t_exp` with a value type of `ST`. Temporary expressions allocate memory based on the type deduction of the rvalue expression, proving a tool for generating real-time expressions and avoiding the costly recomputation of derivatives when a derivative is employed in multiple expressions after a first evaluation.

## 2.3 Scalable time integration with Odeint and OpenFPM

For numerically solving mathematical equations describing the dynamics of a system or process, time-integration methods are necessary. Time-integration methods are numerical algorithms that approximate the trajectory of a dynamical system to a future point in time. Model dynamics is simulated by iteratively taking (sufficiently small) time steps. Methods that, in each iteration, extrapolate the next time step from the current and past ones are called *explicit* time integrators, or *time-stepping* schemes. They successively advance the state $\mathbf{u}$ of the simulation from a time $t$ to a time $t + \delta t$, $\delta t > 0$, given an expression for its time derivative

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathcal{F}(t, \mathbf{u})$$

starting from an initial condition $\mathbf{u}(0) = \mathbf{u}_0$. The right-hand side $\mathcal{F}$ is given by the model that is solved or simulated, or it results from spatial discretization of a partial differential equation (PDE). In order for the thus-computed sequence $\mathbf{u}_0$, $\mathbf{u}_1$, $\mathbf{u}_2$, ... to converge to the true dynamics $\mathbf{u}(t)$ when $\delta t \to 0$, a time-integration method must be both *consistent* and *stable*.

In simulations of nonlinear or stiff dynamics, identifying a consistent and stable time-integration method often requires experimenting with different step sizes and stepping algorithms, or autotuning the choices [110]. This creates additional overhead in parallel simulation codes, because more accurate (i.e., higher order of consistency) or more stable time-integration methods require additional evaluations of the right-hand side $\mathcal{F}$ at intermediate time points (called "stages") [111], which need to be synchronized and communicated between processes. Moreover, since the time-step size is limited by the fastest dynamics to be resolved, methods with uniform step sizes are often wasteful, and instead adaptive methods are used [112] that continuously adjust the step size to the simulated dynamics. This, however, creates additional synchronization and communication overhead in a distributed parallel simulation. In a distributed-memory code, the additional communication needs to be implemented manually, hampering autotuning and experimentation.

In this section, we present a distributed algebra system for integrating the time-stepping library Boost Odeint [113,114] into OpenFPM [115] for scientific computing. We complement the previously described expression system with a distributed template algebra that allows OpenFPM to interface with Odeint. This enables the use of abstract encapsulation of time derivatives in a simulation model, which can directly be used by Odeint for time stepping with a variety of high-order, stiff, and adaptive time-integration methods (see Table 2.1).

### 2.3.1 Related works

Distributed parallel time stepping was available in the discontinued PPM Library [116]. The right-hand side $\mathcal{F}$ was specified by a function pointer and all communication and stage evaluations were encapsulated. Available integrators included explicit Euler with and without super time stepping, 2- and 4-stage Runge–Kutta schemes, Williamson's low-storage third order Runge–Kutta scheme, and 2- and 3-stage TVD Runge–Kutta. Schemes of order $>4$ were not available, nor adaptive time stepping methods. Moreover, PPM was written in Fortran 90/95, which limited the implementation to primitive data types in 2D and 3D. Also the Portable, Extensible Toolkit for Scientific Computation, PETSc [117], offers a variety of time-integration methods, including strong stability-preserving Runge–Kutta methods and their low-storage Implementations. However, it does not support multistep explicit methods, such as the Adams-Bashforth-Moulton predictor-corrector scheme. PETSc's primary focus has been on implicit methods and on handling the overhead of creating a Jacobian matrix in highly optimized C code. This is not straightforward to extend to more complex PDEs. Furthermore, PETSc's GPU support is still under development and

Table 2.1: Explicit time stepping schemes available in Odeint.

| Category | Method | Accuracy order |
|---|---|---|
| fixed step size | explicit Euler | 1 |
| | modified midpoint | 2 |
| | Runge-Kutta 4 | 4 |
| | Runge-Kutta-Cash-Karp 54 | 5 |
| | Runge-Kutta-Dopri 5 | 5 |
| | Runge-Kutta-Fehlberg 78 | 8 |
| dynamic step size | Runge-Kutta-Cash-Karp 54 | 5 |
| | Runge-Kutta-Dopri 5 | 5 |
| | Runge-Kutta-Fehlberg 78 | 8 |
| multi-step | Adams-Bashforth | 1...8 |
| | Adams-Bashforth-Moulton | 1...8 |
| symplectic | symplectic Euler | 1 |
| | velocity Verlet | 2 |
| | Runge-Kutta-McLachlan | 4 |

performance improvements over the CPU code are limited to certain problems and certain problem sizes [118]. The SUNDIALS library [119] provides a range of both implicit and explicit time-integration schemes and supports parallel computing through MPI (Message Passing Interface). However, its focus is on ordinary differential equations (ODEs) and differentiable algebraic equations (DAEs). It is not suitable for time integration of PDE problems that require spatial derivatives as part of the right-hand side $\mathcal{F}$.

Importantly, none of the previously available parallel or distributed time-integration libraries was embedded in an expression language for discretizing spatial derivatives, hence lacking the concept of spatial operators and domain decompositions of particle sets. This forced the user to hard-code any given right-hand side $\mathcal{F}$ and spatial differential operator discretization scheme, leading to lengthy and monolithic code, as well as problem-dependent convergence guarantees of the nonlinear solvers. The algebra system presented here modularly separates the spatial and temporal parts of a PDE solver and enables more compact code.

Beyond distributed computing, several time-integration libraries are available for sequential processors. A widely used example is the library LSODA [120], which is used as a back-end in several popular scientific libraries, including SciPy [121]. LSODA is primarily written in C and provides a high-performance alternative for rapid prototyping of simulation codes that do not require distributed computing. Other examples include CISR-ODE, a solver for code-based system dynamics simulations [122], which is in turn also based on Odeint [113]. However, CISR-ODE is limited to a specific data type, as it

relies on the Armadillo matrix library as a state-type, and it does not support distributed computing. Odeint [113] by itself does support MPI-based distributed data structures, but their use renders the code specific to a certain right-hand side $\mathcal{F}$. The present work bridges this gap, combining modularity of both the data type and the right-hand side with distributed-computing scalability and transparent GPU support.

### 2.3.2 Linking Boost Odeint with OpenFPM

Boost Odeint [114] is a templated C++ library for numerically solving initial value problems modeled by ordinary differential equations (ODEs). It offers a selection of time-stepping methods to numerically solve ODEs, and allows for implementation of custom time steppers. Odeint is developed using meta-programming independent of the underlying data structures, referred to as *state* of a data type *state-type*. The *state* represents the solution of the ODE *System* at a given time. To use Odeint, a *state* has to be provided as an input, along with a *System* function or a functor that computes the right-hand side $\mathcal{F}$ for the current *state* $\mathbf{u}$. One can then use any available time stepper from Odeint, as summarized in Table 1, to extrapolate the *state* to the next time step and update it in-place.

In adaptive time-stepping methods, the size of each step is individually computed from an error estimator and a predefined error tolerance. If the tolerance is violated, the step is rejected and the step size is reduced. This is repeated until the step is accepted. For error estimations below the tolerance, the step size is increased. In Odeint, steppers that are able to approximate the error in the solution are called *error stepper*s. Error-controlled adaptive steppers can be constructed for any *error stepper*. A special class of adaptive steppers are dense steppers, which are internally adaptive but compute the solution at user-defined time points. We refer to the Odeint documentation for more information [114].

Odeint requires a constructor and an *Algebra* for the *state*. The constructor is used for creating temporary variables of the same *state-type* that may be required for a specific stepper. The *Algebra* defines arithmetic operations on the *state-type*, which allows Odeint to compute combinations of *state*s. To use adaptive or error-controlled steppers, the *state* object must also provide a method to compute a norm (i.e., a mathematical "distance") of the corresponding *state-type*. Additionally, another function/functor can be passed to Odeint as an *observer*, which is called before every time step. This can be used to write output or provide additional code to be run at each time step. The interplay of these objects in Odeint is illustrated in Fig. 2.3.

Odeint supports different *state-type*s, like `std::vector` and `boost::array`. None of the supported *state-type*s, however, allow for distributed computing on heterogeneous architectures. User-defined custom *state-type*s can, however, be implemented.

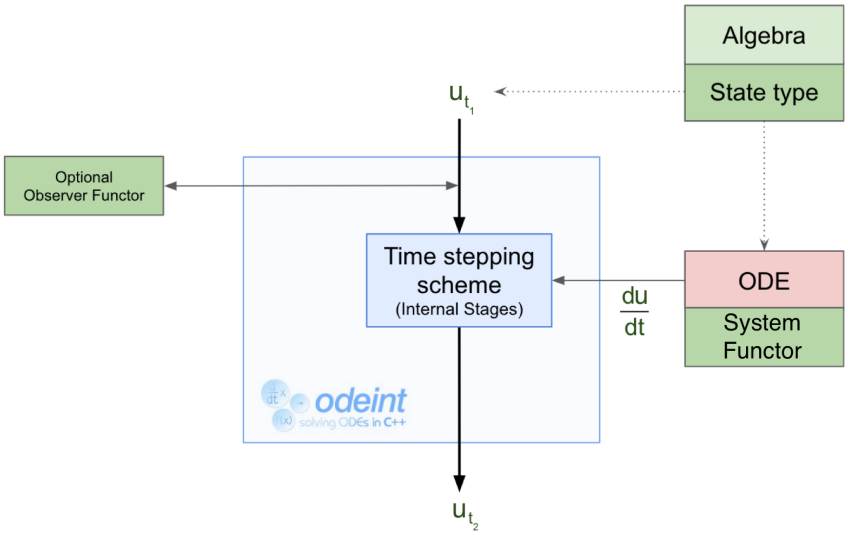



Figure 2.3: Architecture of the Odeint Library. In each iteration, the *state* $\mathbf{u}$ is advanced from a time $t_1$ to a later time $t_2 = t_1 + \delta t$, $\delta t > 0$. The right-hand side $\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathcal{F}(t, \mathbf{u})$ of the ODE system is encapsulated in the *System* functor. An *Algebra* defines mathematical operations over the state type.

### 2.3.3 The distributed OpenFPM state type and algebra

From the architecture of Odeint described above, it becomes clear that it can be extended to distributed and GPU-accelerated computing by providing a custom *state-type* along with its *Algebra*. The OpenFPM platform for scalable scientific computing [115] provides abstract data structures and performance-portable algorithms for shared- and distributed-memory HPC as well as multi-GPU computing implemented in C++ using template meta-programming [123]. It is compatible with Odeint in style and architecture. We therefore propose custom Odeint *state-type*s for OpenFPM objects, which can internally be distributed and specialized for different hardware backends. Specifically, we implement Odeint *state-type*s and *Algebra*s for OpenFPM distributed vectors. These *state-type*s are named `state_type_#d_ofp`, where `#` is substituted with the natural number specifying the dimensionality of the vectors in the array. For example, the *state-type* for a scalar field is `state_type_1d_ofp` and for a 3D vector field `state_type_3d_ofp`. Since OpenFPM types can be internally distributed, the *Algebra* implementation is distributed as well, encapsulating all necessary communication.

OpenFPM also transparently supports GPU computing [123, 124]. For this, we also provide a *state-type* implemented with OpenFPM GPU vectors and an *Algebra* based on GPU kernels (both CUDA for Nvidia GPUs and HIP for AMD GPUs) as `state_type_#d_ofp_gpu`

and `vector_space_algebra_ofp_gpu`, respectively.

```
1  //Templated with OpenFPM distributed vector type
2  template<typename vector_type>
3  struct state_type_2d_ofp{
4    state_type(){}
5    typedef size_t size_type;
6    typedef int is_state_vector;
7    aggregate<texp_v<double>,texp_v<double>> data;
8
9    //Method to get the size
10   size_t size() const
11   { return data.get<0>().size(); }
12
13   //Method to resize
14   void resize(size_t n)
15   {
16     data.get<0>().resize(n);
17     data.get<1>().resize(n);
18   }
19 };
20 //Additional structs as required by Odeint
21 namespace boost::numeric::odeint {
22   template<>
23   struct is_resizeable<state_type<vector_type>> {
24     typedef boost::true_type type;
25     static const bool value = type::value;
26   };
27   template<typename T>
28   struct vector_space_norm_inf<state_type<T>>
29   {
30     typedef typename T::stype result_type;
31   };
32 }
```

Listing 2.7: The OpenFPM distributed vector *state-type* for Odeint.

Listing 2.7 shows as an example the *state-type* for OpenFPM distributed vector with two components of type `double` expressed as an `aggregate` (Line 7). The container `texp_v` is a special transient expression as described in the previous section. It allows array and tensor operations to be expressed using MATLAB [125] or Numpy [126] syntax [64]. It also supports implicit mathematical equations that require matrix assembly and numerical solution of a system of algebraic equations. It is therefore a convenient choice for storing (temporary) results in a way that is compatible with the operators of the embedded DSL, except operations that require inter-process communication [64]. Because `texp_v` is a resizable object, we notify Boost that the entire *state-type* is resizable (Lines 22–25). In addition, since adaptive steppers require calculating a norm of the *state*, we specify the result type of the infinity norm (Line 28).

To endow Odeint with support for the OpenFPM *state-type*s, we define a custom *Algebra*

in the scope of odeint called `odeint::vector_space_algebra_ofp`. This vector algebra is made available by the header file `OdeIntegrators/vector_algebra_ofp.hpp` and defines the distributed mathematical operations on OpenFPM *state-type* objects. The algebra needs to be specified at compile time (cf. Listing 2.9, Line 6).

Our distributed *Algebra* defines custom `for_each#()` and `for_each_prop#()` inline methods using OpenFPM's distributed iterators that encapsulate the algebraic operations on the `texp_v` container for multidimensional fields. The *Algebra* `vector_space_algebra_ofp` consists of a total of `#`=15 inline methods to support all available Odeint time steppers. Further, it provides the methods `for_each_norm()` and `for_each_prop_resize()` for computing *state* norms in adaptive stepping and for distributed *state* object resizing, respectively. Code is generated at compile time from this *Algebra* in conjunction with the OpenFPM template expression system for differential equations [64].

### 2.3.4 Implementation of models for scalable time integration

With the OpenFPM-based distribute *state-type*s and *Algebra* described above, the user can implement ODE models of dynamical systems to be simulated using our expression system as described in section 2.2, with any of the time integrators provided by Odeint to express time derivatives. The performance portability and scalability of the OpenFPM data structures is inherited.

In order to implement an ODE model, the right-hand side $\mathcal{F}$ needs to be provided as an Odeint *System* function or a functor with the signature of the *state* class. This function or functor is called by the time stepper at each stage. The *System* object is parameterized by the templated *state-type* and the type of its time derivative. Although it is unlikely that the solution state and its time derivative have different *state-type*s, Odeint allows them to be specified separately in order to provide the most general framework possible. Finally, the current time $t$ is a parameter to the *System*. Using these parameters, the *System* computes $\mathcal{F}(t, \mathbf{u})$ based on the current state $\mathbf{u}_t$ at time $t$.

```
1  //Templated type of the time derivative operator
2  template<typename Laplacian_type>
3  struct System_Functor
4  {
5    Laplacian_type &Lap
6    double K = 0.053, F = 0.014, d1 = 2e−4, d2 = 1e−4; //Physical constants
7
8    System_Functor(Laplacian_type &Lap) : Lap(Lap)
9    {}
10
11   void operator()(const state_type_2d_ofp &u, state_type_2d_ofp &dudt,
12   const double t ) const
13   {
14     //Get OpenFPM distributed domain
15     ofp_dist_vector_type &Domain= *(ofp_dist_vector_type*) DistVecPointer;
16
```

```
17      auto C=getV<Conc>(Domain) ; //Get the property
18
19      //Get the data from the current state.
20      C[0]=u.data.get<0>();
21      C[1]=u.data.get<1>();
22
23      //Compute the time derivate using PDE expressions.
24      Domain.ghost_get<Conc>();
25      dudt.data.get<0>() = d1*Lap(C[0]) - C[0] * C[1] * C[1] + F - F * C[0];
26      dudt.data.get<1>() = d2*Lap(C[1]) + C[0] * C[1] * C[1] - (F+K) * C[1];
27    }
28  };
```

Listing 2.8: Example *System* functor for the 3D Gray-Scott problem Eq. (2.2) in OpenFPM.

Listing 2.8 exemplifies this using the *System* functor of the 3D Gray-Scott reaction-diffusion problem from Sec. 2.5. As evident from the example, it is straightforward to integrate templated OpenFPM derivatives operators for spatial derivatives [64] into the ODE system. The *System* functor can also contain any additional code that is to be run for a right-hand-side evaluation, for example for performance profiling. Since the data as well as the operations can be distributed by OpenFPM in a multi-node or multi-GPU computer, boundary layers ("ghost layers") may need to be communicate between processes. This ghost communication is implemented in the OpenFPM `Domain` class. A reference to the OpenFPM `Domain` object is therefore obtained in Line 15 of the listing from a pointer to the OpenFPM distributed domain structure. Data from the *state* is then stored in the property of the OpenFPM `Domain` compatible with the spatial operators computed. The inter-process communication for the `ghost_get()` is then executed during right-hand-side evaluation of each ODE stage right from within the *System* functor (Line 24), guaranteeing consistent *state*s to Odeint at all times.

After defining the *System*, any Odeint time stepper can be used for time integration in the `main()` program, as shown in Listing 2.9 for the *System* from Listing 2.8.

```
1  //Initialize OpenFPM data strucures and spatial derivatives
2  auto C=getV<Conc>(Domain);
3  Laplacian Lap(Domain);
4
5  //Define stepper type
6  odeint::runge_kutta4<state_type_2d_ofp, double, state_type_2d_ofp, double, odeint::
       vector_space_algebra_ofp> Odeint_rk4;
7
8  //Declare and initialize state
9  state_type_2d_ofp u;
10 u.data.get<0>()=C[0];
11 u.data.get<1>()=C[1];
12
13 //Initialize System functor with spatial Laplacian R.H.S.
14 System_Functor<Laplacian> System(Lap);
15
```

```
16  //Invoke RK4 stepper for t=[0,20] with time step 0.1
17  double t =0, tf = 20, dt = 0.1;
18  size_t steps = integrate_const(rk4(), System, u, t, tf, dt);
```

Listing 2.9: An OpenFPM program using an Odeint time stepper.

The stepper used in Line 6 is one of the built-in steppers of Odeint (see Table 2.1). Before starting time integration, we initialize the *state* in Lines 10–12 using the 2D OpenFPM vector state type `state_type_2d_ofp`. The two components of the vector are then set. The correct distributed *Algebra* for this state type is instantiated in Line 6 from `odeint::vector_space_algebra_ofp` as described above. We then initialize the system functor from Listing 2.8 in line 14. Then, the time stepper can be invoked to run until a final time as shown in line 18. Alternatively, the method `do_step()` of the stepper can be called to perform a single time step only. Time step adaptation with error-controlled adaptive steppers can be done by calling `integrate_adaptive()` with the desired tolerance. The stepper returns the number time steps performed, and the state is updated in-place. Importantly, and thanks to the use of OpenFPM, this program runs on shared- and distributed-memory CPU systems, as well as on GPUs and mutli-GPU computers.

## 2.4 Accuracy benchmarks

We benchmark the implementation for correctness on problems with known analytical solution. In order to generate many such problems, we consider the parametric family of exponential dynamics and sigmoidal dynamics defined by:

$$f(x,y,t) = A(x,y)\mathrm{e}^t \qquad \frac{\partial f(t)}{\partial t} = f(x,y,t), \quad (x,y) \in [0,1]\times[0,1], \tag{2.1a}$$

$$f(t) = \frac{1}{1+\mathrm{e}^{-t}} \qquad \frac{\mathrm{d}f(t)}{\mathrm{d}t} = f(t)(1-f(t)). \tag{2.1b}$$

We choose $A(x,y) = xy$ as the function that depends on the spatial location. Hence, the exponential family contains as many different ODEs as the number of discretization points in space. We use the OpenFPM implementation of Odeint to solve this problem with the analytical solution as initial condition at $t_0 = -5$. We compare the computed solution with the analytical solution at final time $t_f = 5$ and compute the infinity norm $\|\mathbf{e}\|_\infty = \max(|e_1|,\ldots,|e_n|)$ and the Euclidean $L_2$ norm $\|\mathbf{e}\|_2 = \sqrt{e_1^2+\ldots+e_n^2}$ of the absolute error $\mathbf{e} = \mathbf{u} - \mathbf{u}_{\text{exact}}$ across all $n$ time steps. Methods with fixed time-step size are tested for different $\delta t$ to verify the order of convergence.

We first test the fixed-step multi-stage methods Runge-Kutta 4, Runge-Kutta-Dopri 5, and Runge-Kutta-Fehlberg 78 (see Table 2.1) for the exponential dynamics in Eq. (2.1a). Figure 2.4a shows the convergence plots for increasing numbers of time steps. All methods converge with expected order as indicated by the solid lines until machine precision is reached and finite-precision roundoff errors start to accumulate. The same is observed in Fig. 2.4b

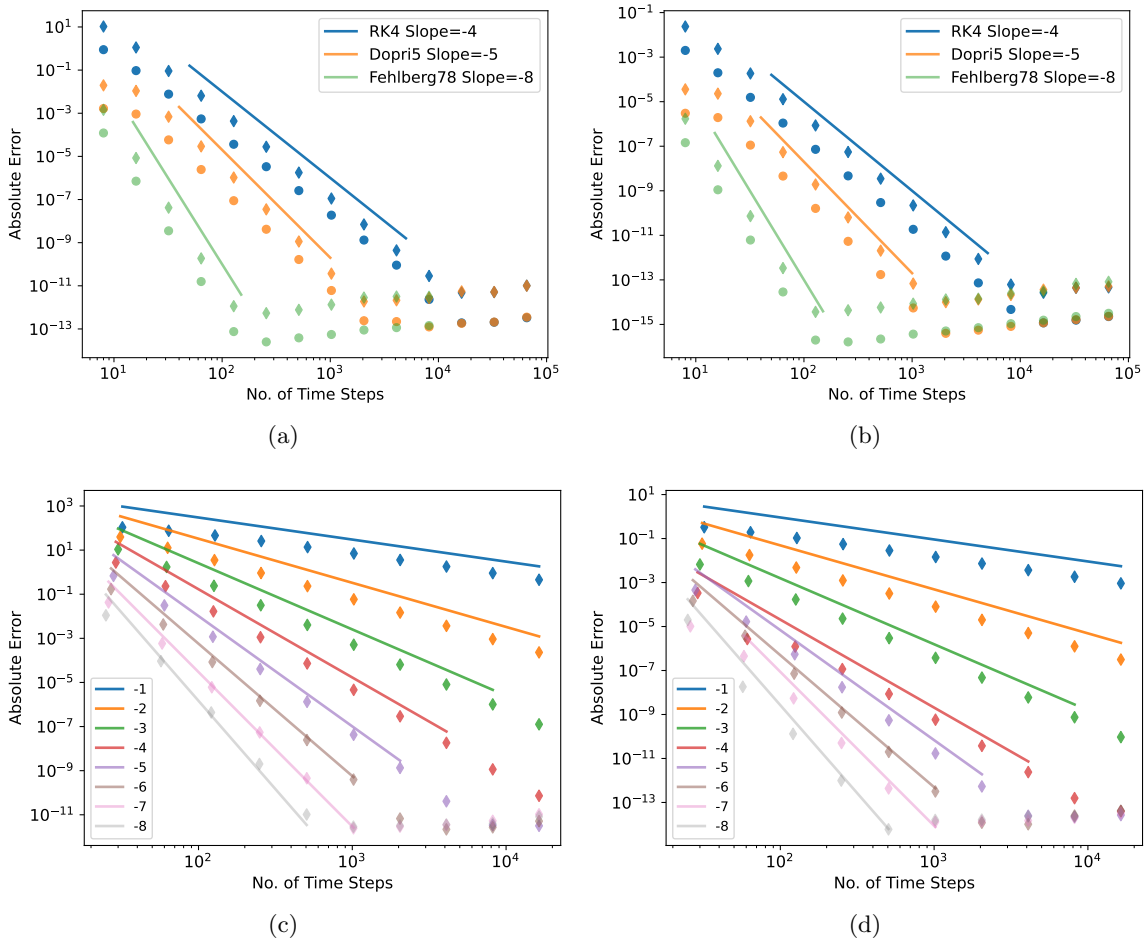



Figure 2.4: **Numerical convergence of time integration schemes with increasing numbers of time steps. (a,b)** $L_\infty(\blacklozenge)$ and $L_2(\bullet)$ error norms for the exponential dynamics of Eq. (2.1a) (**a**) and the sigmoidal dynamics of Eq. (2.1b) (**b**) with different one-step multi-stage methods (colors, inset legend). Solid lines indicate the theoretically expected slopes. **(c,d)** $L_\infty(\blacklozenge)$ error norms for the exponential (**c**) and sigmoidal (**d**) dynamics solved using Adamsh-Bashforth with different numbers of steps ($1 \ldots 8$, colors, inset legend). Solid lines indicate the theoretically expected slopes.

for the sigmoidal dynamics of Eq. (2.1b). Next, we test the fixed-step Adams-Bashforth implementation as an example of a multi-step method. We solve the same problems using 1 to 8 previous steps, resulting in convergence orders between 1 and 8. This is verified for the exponential and sigmoidal dynamics in Figs. 2.4c and 2.4d, respectively. In all cases, the theoretically optimal order of convergence is observed down to machine precision.

All tests are repeated for different numbers of OpenFPM processes (1, 2, 6, 24) to confirm that the results are the same, regardless of the degree of parallelism. Taken together, this confirms the correctness of the implementation of our Odeint *state-type* and *Algebra* for distributed OpenFPM data types.

## 2.5 Scalability benchmarks

We test the scalability of the present implementation for multi-stage methods, which incur additional communication overhead in a distributed-memory parallel program when the ODEs result from spatial discretization. We again test the fixed-step time integrators Runge-Kutta 4, -Dopri 5, and -Fehlberg 78, and additionally -Dopri 5 with adaptive time steps. As a baseline, we use a native implementation of Runge-Kutta 4 in OpenFPM without using Odeint and the associated *Algebra*. We solve the exponential dynamics described in Eq. (2.1a) on $512 \times 512$ points that discretize the domain $(x, y) \in [0, 1]^2$ with equal spacing. This amounts to $262,144$ different ODEs from the exponential family who are distributed amongst the processes using OpenFPM domain decomposition and solved in parallel. We perform a strong scaling measurement with constant problem size distributed across increasing numbers of processes (1 process per CPU core). The measurements are taken on a cluster of Intel Xeon E5-2680v3 CPUs with each node containing 24 ($2\times12$) cores with shared memory. The nodes in the cluster are connected using a 4-lane FDR InfiniBand network (at 14 Gb/s per lane) with a latency of 0.7 µs for message passing using the OpenMPI library.

The results are shown in Fig. 2.5a. The native OpenFPM implementation is about an order of magnitude slower than the Odeint+OpenFPM implementation. This is likely due to the performance gains from optimizing the on-the-fly computation of stages in Odeint as described [114]. Since there is no spatial coupling in this problem, we find close to ideal (solid black line) scaling in computational time for all tested steppers.

The present software also natively supports parallel computing on GPUs of Nvidia (via CUDA) and AMD (via HIP). We therefore compare the speedup achieved when running the same exponential dynamics simulation on a Nvidia consumer GPU, a RTX 4090, over using all cores of an AMD Ryzen 3990X CPU (=1.0). The results are shown in Fig. 2.5b. Depending on the problem size (spatial resolution and time-step size), the same code is about 2 times faster when run on the GPU.

To test how this changes when spatial coupling and corresponding communication overhead is introduced, we consider the 3D Gray-Scott reaction-diffusion problem that couples the ODEs in space:

$$\frac{\partial}{\partial t}\begin{bmatrix} C_0(t,x,y,z) \\ C_1(t,x,y,z) \end{bmatrix} = \begin{bmatrix} d_1 \Delta_{(x,y,z)} C_0 - C_0 C_1^2 + F(1-C_0) \\ d_2 \Delta_{(x,y,z)} C_1 + C_0 C_1^1 - (F+K) C_0 \end{bmatrix} \tag{2.2}$$

for the parameters $d_1 = 2 \cdot 10^{-4}$, $d_2 = 10^{-4}$, $F = 0.053$, and $K = 0.014$. The Laplace

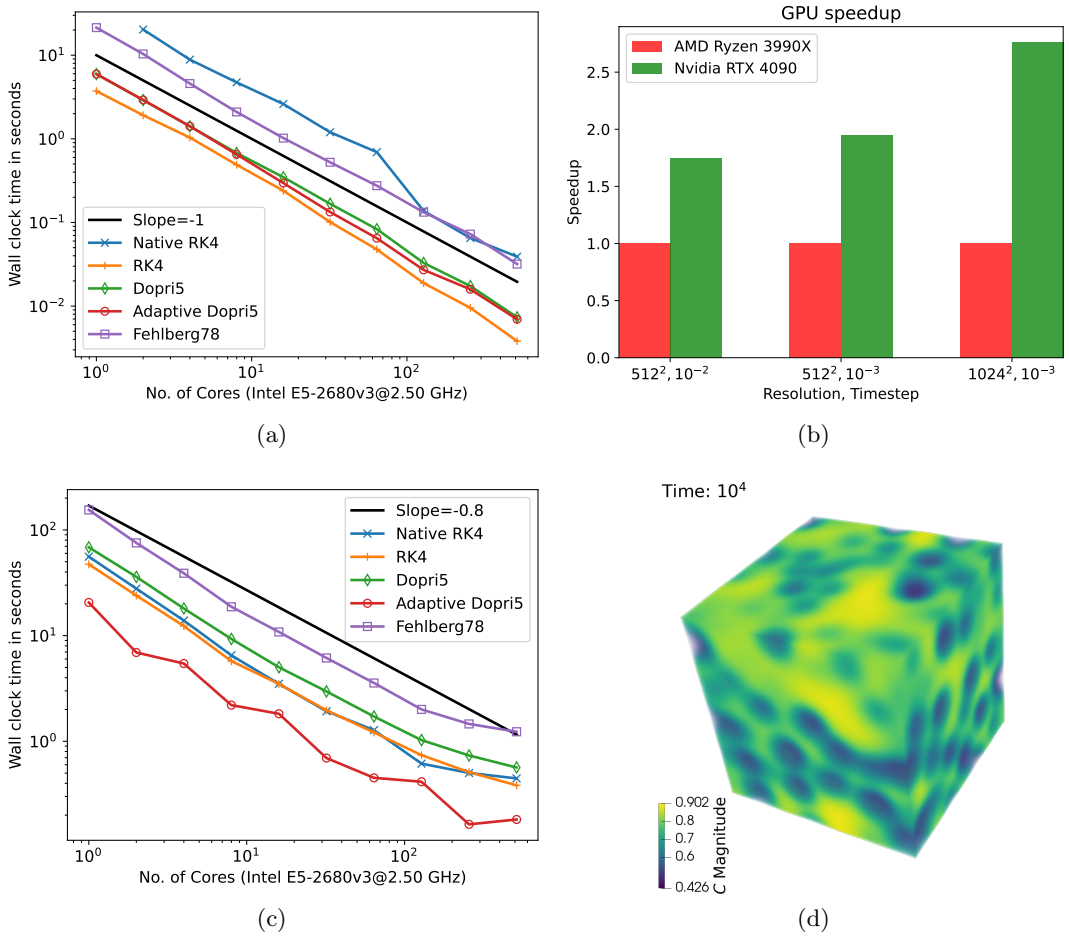



Figure 2.5: **Strong scaling of the OpenFPM+Odeint time integration schemes with increasing numbers of CPU cores.** **(a)** Average Wall-clock time in seconds over 3 runs for solving the exponential dynamics from Eq. (2.1a) using different one-step multi-stage methods (colors, inset legend, error bars below symbol size). **(b)** GPU speedup for the exponential dynamics simulated by RK4 for different resolution and time step sizes when compared to the single core run. **(c)** Average Wall-clock time in seconds over 3 runs for the 3D Gray-Scott Eq. (2.2) using different one-step multi-stage methods (colors, inset legend). **(d)** Visualization of the simulated Gray-Scott reaction diffusion model state at $t = 2 \cdot 10^4$.

operators in space $\Delta_{(x,y,z)}$ are discretized using the DC-PSE method [127] as implemented in the PDE template expression system of OpenFPM [64]. We solve the Gray-Scott model in the 3D cube $[0, 2.5]^3$ with periodic boundary conditions on a regular Cartesian grid

of $64 \times 64 \times 64$ points, time stepping with $\delta t = 1$ until final time $t_f = 20$. The wall clock times for different multi-stage schemes are plotted in Fig. 2.5b. We find a parallel efficiency of nearly 80% on 512 CPU cores for all tested multistage schemes with inter-processor communication of stages and of spatial differential operators performed inside the *System* functor. The native OpenFPM RK4 implementation has similar performance as the OpenFPM+Odeint implementation, suggesting the bottleneck to be the spatial derivative computation. Also, adaptive time stepping does not incur significant scalability penalty. The initial condition at $t = 0$ and the simulated state at $t = 10^4$ are visualized in Figs. 2.5d, showing the self-organized emergence of the characteristic Turing pattern [128–130] of this model.

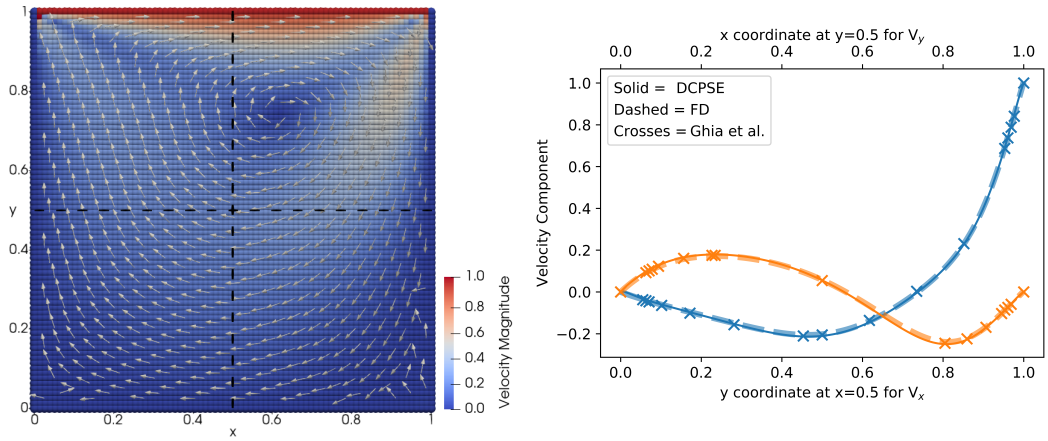
## 2.6 Further applications

We test our implementation and demonstrate the use of the presented expression system on two benchmark problems from biological hydrodynamics. First, we consider the incompressible Navier-Stokes equations describing fluid flow at larger length scales where inertial forces play a role. Examples include air flow in the lungs [131] and blood flow in the heart [132, 133]. As a showcase, we consider the nonlinear lid-driven cavity problem, which is a classic benchmark to check a simulation code's ability to solve for steady-state Navier-Stokes flow. As an algorithmic novelty, we use our expression system to implement a mesh-free simulation with pressure correction. We validate the solver by comparing against simulation data from Ghia et al. [16]. Further, we use this test case to evaluate how many lines of code need to be edited in order to change the mesh-free simulation to a finite-difference simulation on a regular Cartesian grid.

Second, we adapt the mesh-free solver with incompressible pressure correction to simulating the time-resolved dynamics of active polar fluids in two dimensions. These coupled PDEs describe the mechanics of active biological materials, such as the acto-myosin cortex in cells [27, 134], by self-organized mechano-chemical deformation [62]. We use this test case to demonstrate how to use our expression system to discretize PDEs in a Lagrangian frame of reference. We validate the resulting simulation code in a convergence study showing the correct scaling of the numerical error. We further use this second test case to benchmark the scalability of the resulting OpenFPM simulation code on a multi-core computer, illustrating the ease of writing the code that runs on multiple CPU cores.

### 2.6.1 Incompressible Navier-Stokes - Lid driven cavity

As a first test case we consider the nonlinear lid-driven cavity problem governed by the incompressible Navier-Stokes equations in the unit square $[0, 1]^2$ with the top boundary (i.e., the "lid") moving at constant velocity $\mathbf{v}_b = (1, 0)^\top$ and the rest of the boundaries




(a) (b)

Figure 2.6: Nonlinear lid-driven cavity problem of Reynolds number $\mathrm{Re} = 100$ solved on an $81 \times 81$ grid using the pressure-correction algorithm. **(a)** Visualization of the velocity magnitude (color) and direction (arrows) as computed by the implicit DC-PSE solver in the two-dimensional simulation domain. **(b)** Velocity $x$-component (blue) along the vertical line $x = 0.5$ (dashed in **a**) and $y$-component (orange) along the horizontal line $y = 0.5$. We compare the present solution computed using second-order DC-PSE (solid lines) or finite differences (FD, dashed lines) with the available reference data from Ghia et al. [16] (crosses) for the same Reynolds number.

having no-slip boundary conditions. The governing equations are:

$$\mathbf{v} \cdot (\nabla \mathbf{v}) - \frac{1}{\mathrm{Re}} \Delta \mathbf{v} = \nabla \Pi \tag{2.3a}$$

$$\nabla \cdot \mathbf{v} = 0 \tag{2.3b}$$

$$\mathbf{v}(x_b, y_b) = (0, 0), \text{ except } \mathbf{v}(x_b, 1) = (1, 0)\,, \tag{2.3c}$$

where $x_b$, $y_b$ are the coordinates of the boundary, $\Pi$ is the pressure, and Re is the Reynolds number. We numerically solve these equations in primitive variables, velocity and pressure, for Reynolds number $\mathrm{Re} = 100$. The incompressibility condition in Eq. (2.3b) is imposed using a pressure-correction scheme [135]. We discretize the differential operators in space using Discretization-Corrected Particle Strength Exchange (DC-PSE) [77] of second order convergence. DC-PSE is a generalization of finite-difference methods and has been previously used to solve Navier-Stokes problems using velocity-vorticity correction [78, 136]. However, to our knowledge, it has never been used in a pressure correction algorithm.

Pressure correction [135] is a method from computational fluid dynamics to impose the incompressibility condition in numerical simulations of the incompressible Navier-Stokes

equations. Due to the nonlinearity of the convection term, numerical solutions generally violate the incompressible continuity Eq. (2.3b), i.e., the flow velocity field is not guaranteed to be divergence-free. The idea of pressure correction is to use the pressure field as a Lagrange multiplier, which is determined such that the velocity becomes exactly divergence-free. This algorithm is described in detail in the next chapter for steady-state solutions of 3D active fluids. In essence, it iteratively solves for the velocity field with a given pressure field, then computes the correction potential for incompressibility, corrects the velocity and the pressure accordingly, and iterates until convergence. This simulation algorithm has previously been used in conjunction with moving least squares discretization methods [137].

We discretize the square 2D domain with collocation points arranged on a regular Cartesian lattice of $81 \times 81$ points. We use DC-PSE operators of convergence order 2 with an interaction cutoff of 3.1 grid cells. We implement this algorithm in our C++ expression system, which requires 156 lines of code, not counting comments and empty lines. A visualization of the simulation result is shown in Fig. 2.6a. To validate the simulation, we compare with simulation data from Ghia et al. [16] for the $x$-component of the velocity along a line across the domain in the $y$-direction at $x = 0.5$, see Fig. 2.6b.

We further use this test case to quantify how difficult it is to change the simulation to using a different numerical method. Therefore, we edit the code to use finite-difference stencils instead of DC-PSE. This requires deleting the boundary particle detection, changing the declaration of the differential operators, and including the finite-difference OpenFPM library. Altogether, it requires changing 21 lines of code. The overall simulation logic and all PDE expressions remain unchanged with the solution still matching the benchmark data (see Fig. 2.6b, dashed line). This illustrates how the present expression system could accelerate testing different numerical methods in a scalable parallel simulation program.

### 2.6.2 Two dimensional active fluid

In the second test case we implement a solver for the active polar fluid equations in two dimensions and compare with a benchmark simulation [17]. The nonlinear, non-equilibrium hydrodynamics of active polar fluids described by set of PDEs that are well known [21, 22], and will be discussed in the next chapter Eq. (3.1). These equations describe a non-equilibrium, nonlinear incompressible Stokes flow problem with 4 unknown fields (the scalar field $\Pi$, the vector fields $\mathbf{v}$ and $\mathbf{p}$, and the tensor field $\sigma$) and 11 terms on the right-hand side altogether as shown in Appendix B.1. For PDEs of such complexity, our expression system is particularly useful, as it makes the simulation code more readable and its implementation less error-prone.

We discretize all fields on Lagrangian particles that move with the velocity $\mathbf{v}$ of the flow. Differential operators are consistently approximated over the irregularly distributed set of moving collocation points using DC-PSE [77]. This enables an important difference to previous approaches [17], which is that we do not need to interpolate the Lagrangian particles to a regular grid at every time step. Instead, we solve the implicit equation for the

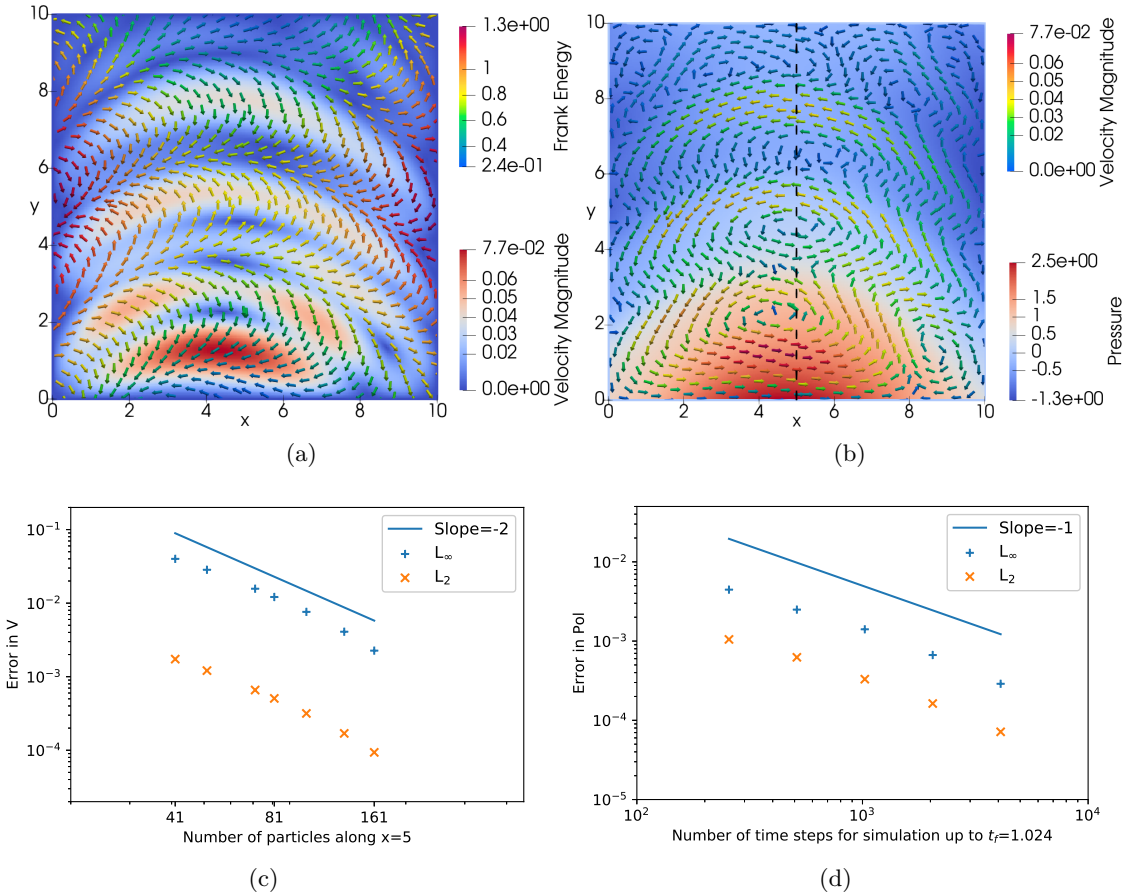
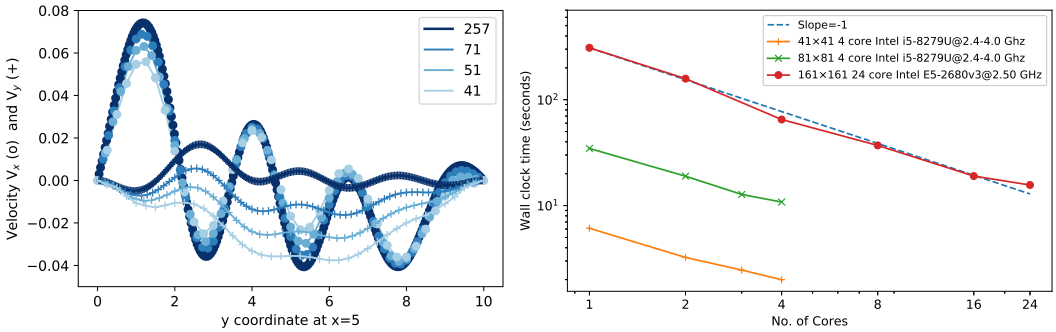



Figure 2.7: Visualization and grid-convergence of the velocity and polarity fields for the two-dimensional viscous active polar fluid simulation. **(a)** The polarity field (arrows colored by Frank free energy) and of magnitude of the velocity (background color). **(b)** Velocity field (arrows colored by magnitude) and of the pressure (background color). **(c)** Spatial grid-convergence of the error in the velocity vectors compared to a highly resolved simulation of $257 \times 257$ particles. Both the $L_2$ and $L_\infty$ norms of the absolute error at $t = 2 \cdot 10^{-6}$ over all particles are shown. **(d)** Temporal grid-convergence of the error in the polarity vectors for an increasing number of time steps to reach time $t = 1.024$, compared against a highly resolved simulation with 16 384 time steps on $41 \times 41$ particles. The solid lines in **c** and **d** show the theoretically expected error scaling.

velocity directly on the irregularly distributed particles, using DC-PSE operators to build the system matrix and the KSPGMRES linear system solver from PETSc [98] to solve the system.




(a) (b)

Figure 2.8: **(a)** Profile of the $x$- ($\bullet$) and $y$-components ($+$) of the velocity along the black dashed line in Fig. 2.7(a) at $x = 5$ for initial grids of increasing resolution from $41 \times 41$ to $257 \times 257$ (grayscale, see inset legend) showing convergence of the solution. **(b)** Strong scaling of the active polar fluid simulation for different initial grid resolutions. We plot the wall-clock time taken to complete 10 simulation time steps on an increasing number of CPU cores for two different processor models. The ideal scaling is indicated by the dashed blue line.

We also extend the simulation to solve for the time dynamics of the fields using an explicit Runge-Kutta time stepping method of order 4 for the polarity field and explicit Euler time stepping for moving the particles., both with time step size $\delta t = 2 \cdot 10^{-7}$. Using the present expression system, the entire simulation code is 340 lines long, reusing 50 lines from the lid-driven cavity solver. Further, by just changing 20 lines of the code, the numerical method can be changed from a DC-PSE discretization on moving Lagrangian particles to a finite-difference simulation on a static Cartesian grid.

We verify our implementation for the benchmark problem from Ref. [17], where a staggered-grid finite-difference scheme has been used in conjunction with Lagrangian particles and remeshing. We therefore solve the governing equations in the square domain $[0, 10]^2$ with edge lengths $L_x = L_y = 10$ and the initial condition for the polarity

$$\mathbf{p}(x, y, 0) = \begin{pmatrix} \sin\left(2\pi(\cos(\frac{2x-L_x}{L_x}) - \sin(\frac{2x-L_x}{L_x}))\right) \\ \cos\left(2\pi(\cos(\frac{2y-L_y}{L_y}) - \sin(\frac{2y-L_y}{L_y}))\right) \end{pmatrix} \tag{2.4}$$

and boundary conditions for polarity and velocity

$$\mathbf{p}(x_b, y_b, t) = \mathbf{p}(x_b, y_b, 0), \tag{2.5}$$

$$\mathbf{v}(x_b, y_b, t) = (0, 0)^\top. \tag{2.6}$$

The simulation proceeds by solving for the steady-state velocity using the given polarity at the initial time. The collocation points (particles) are initially placed on a regular Cartesian grid. From there, the particles are advected by the flow velocity, and the polarity field is evolved according to the Lagrangian derivative in Eq. (3.1a). Since the particles move, DC-PSE kernel weights are recomputed at each time step, and all steps are repeated until the final time. For this simulation, we expect first-order convergence in time, limited by the Euler method used to move the particles, and second-order convergence in space given by the order of the DC-PSE operators. This is confirmed in Fig. 2.7c-d.

We also use this test case to demonstrate that the OpenFPM code generated by our template expression system is fully parallel and scales well—as benchmarked elsewhere [102]—on multi-core computers. For this, we perform a strong scaling experiment of the present active polar fluid simulation for initial grids of different sizes, but without rewriting or manually tuning any of the code. The result in Fig. 2.8b shows a parallel efficiency of 87% when scaling the code up to all 24 cores of an Intel Xeon E5-2680v3 processor at 2.5 GHz clock frequency. Further, on a 4-core Intel i5 mobile consumer CPU, we measure a parallel efficiency of 70% (strong scaling) even when clock boost is enabled for single-core tasks.

## 2.7 Conclusion

We have devised a comprehensive and efficient software framework for numerically solving partial differential equations (PDEs) on high-performance distributed computing architectures. Our work is particularly relevant in hydrodynamics, that combines the benefits of a C++ expression system with a custom state type and distributed algebra system, implemented using OpenFPM, a parallel computing library. The C++ template expression system cleanly separates the definition of the model to be simulated from the implementation of the numerical methods used, allowing new solvers or models to be implemented independently, thereby improving developer productivity significantly.

In this chapter, we described a custom state type and distributed algebra system extends the use of Boost Odeint library of time-integration methods to distributed-memory CPU and GPU HPC systems, thereby broadening its application in distributed numerical simulations. The system is based on the transparently distributed OpenFPM data structures and is compatible with the generic stepper algebra of Odeint, making all explicit time-stepping schemes available. This offers enhanced coding efficiency for new models and reduces the risk of errors.

We illustrated the utility and efficiency of the system in various test cases typical for biological hydrodynamics simulations. This included implementing solvers for incompressible Navier-Stokes, active polar fluids using a Lagrangian particle method, using a novel mesh-free pressure-correction algorithm and finite differences on a regular Cartesian grid. Our experiments demonstrated high parallel efficiency in the test cases.

However, the system is not without its limitations. It is currently compatible only with

numerical methods already implemented in the OpenFPM numerics library. While this encompasses several useful libraries, there are many more that could be added in the future. Additionally, our implementation requires more time to compile compared to a regular C++ code, with the compilation overhead dependent on the complexity and length of the expressions. The compile time error message for an incorrect/incompatible expression is harder to comprehend as the resulting type of the expression can be nested and long leading to a bloated error message of type mismatch. The expressive and compact nature of the developed system somewhat alleviates this problem because the compiler reports the line number that makes checking the expression straightforward by the user. Furthermore, our expression system also necessitates careful selection and tuning of the numerical methods and does not perform auto-tuning. However, we believe it provides an excellent platform for higher-level languages and problem-solving environments.

The presented system renders it straightforward to extend scientific problem solving environments, such as the OpenPME development environment [138, 139], with adaptive and high-order time stepping. Further, it has the potential of extending autotuning systems for spatial discretization methods [110] to the time dimension, as changing the time stepper, or the step size, amounts to a single code-line change that is easily implemented in an autotuning framework. Although our distributed algebra system with Odeint is designed for explicit solvers, IMEX (Implicit-Explicit) schemes that require some part of the equation to be handled implicitly, could be implemented using the optional observer functor. The optional observer functor (see Fig. 2.3) allows for reading and modifying the time-step state before taking an explicit step. Hence, we can handle the implicit part using our implicit solvers described in section 2.2.3. Indeed, we solved the Stokes flow part of the active hydrodynamic equations using the optional observer in the Lagrangian and hybrid particle-mesh settings. Taken together, the presented system allows for rapid rewriting of computational models in parallel CPU and GPU codes with minimal changes to the source code while maintaining scalability.

Future plans include expanding the presented expression system to accommodate additional numerical methods and more general meshes. We also plan to enhance the user-friendliness and portability of the system by providing wrappers for scripting languages and integrating with domain-specific simulation languages [140]. As of now, our C++ template expression system is openly available as part of the OpenFPM library, offering researchers a flexible and efficient tool for numerically solving PDEs on parallel computers.

# Chapter 3

# Numerical schemes for solving bulk active hydrodynamics and surface PDEs

## 3.1 Introduction

In this chapter, we introduce an algorithm to simulate the bulk dynamics of the symmetry-preserving 3D active Ericksen-Leslie equations that model active fluids. Similar to a previous algorithm in two dimensions [17], we present a hybrid particle-mesh approach to study active fluids in 3D. In our approach, we use Discretization Corrected Particle Strength Exchange (DC-PSE) [141] to discretize differential operators, enabling the use of irregular particle distributions, and couple it with a pressure correction scheme to enforce incompressibility. Hence, avoiding the use of staggered grids that are challenging for implementing boundary conditions in complex domains. We show convergence of our scalable iterative pressure-correction solver for the incompressible force balance on regular and irregular particle distributions. To handle the computational cost associated with 3D problems, we use the open-source framework OpenFPM [142]. The length of the equations is tackled with the template expression system as described in the previous chapter, which makes writing the simulation code tractable [143]. We present the derivation of the governing hydrodynamic equations in 3D, a detailed description of the algorithm, validation, and its performance.

Following the introduction to the algorithm for simulating active fluids and the in-depth exploration of a hybrid particle-mesh approach, we consider the implications for curved surfaces. Solving partial differential equations (PDEs) on curved surfaces presents its own unique set of challenges. The complex shapes of biological surfaces demands a solution that can not only handle the irregularities, but also calculate extrinsic differential-geometric quantities with a high order of accuracy. This becomes particularly crucial when dealing with simulations that involve a deforming geometry, where the deformation is governed

by surface force balances or an equivalent surface PDE. Existing methodologies, though effective in handling problems within their scope, often fall short when it comes to efficiently processing complex 3D surfaces due to discretization of the higher-dimensional embedding field. Therefore, there exists a need for an approach that combines the computational efficiency of low-dimensional mesh-free methods with the flexibility of embedding techniques.

It is within this context that we present a collocation method for solving PDEs on smooth, curved surfaces without the need for a mesh. This method employs tracer points situated on the surface to store surface-related values, thus also serving as a means of discretization. While the method evaluates the surface differential operators without an embedding, it uses embedding techniques for the derivation of the operators. The approximations of the operators at the surface tracer points are calculated in the embedding space, and reduced to the surface via projection along the local normal vector. Conceptually, this amounts to projecting discrete operators instead of flux vectors, as is commonly done in embedding methods. The resulting methodology combines the benefits of moving-frame techniques, such as low dimensionality, reduced computational expense, and the elimination of meshes, the advantages of embedding techniques, including their adaptability to complex surfaces, as well as their capacity to calculate extrinsic differential-geometric quantities. This new approach holds the potential to enhance our comprehension and computation of complex geometric problems.

## 3.2 Three dimensional active hydrodynamics

To model the complex dynamics and the emergent phenomena of microtubule bundles with Kinesin-1 motors, we consider a symmetry-preserving active polar fluid model. The Ericksen-Leslie model of three dimensional active fluids can be described by a system of coupled non-linear PDEs [22]. Using Einstein's index summation convention, wherein repeated indices imply summation, the governing equations can be written as:

$$\frac{\mathrm{D}p_\alpha}{\mathrm{D}t} = \frac{h_\alpha}{\gamma} - \nu u_{\alpha\beta} p_\beta + \lambda \Delta\mu p_\alpha \tag{3.1a}$$

$$\partial_\beta \sigma^{(\mathrm{tot})}_{\alpha\beta} - \partial_\alpha \Pi = 0,\ \partial_\gamma v_\gamma = 0 \tag{3.1b}$$

$$\begin{aligned} 2\eta u_{\alpha\beta} = \sigma^{(s)}_{\alpha\beta} + \zeta \Delta\mu \left( p_\alpha p_\beta - \frac{1}{3} p_\gamma p_\gamma \delta_{\alpha\beta} \right) \\ - \frac{\nu}{2} \left( p_\alpha h_\beta + p_\beta h_\alpha - \frac{2}{3} p_\gamma h_\gamma \delta_{\alpha\beta} \right), \end{aligned} \tag{3.1c}$$

with $\alpha, \beta, \gamma^1 \in \{x, y, z\}$ for the spatial components denoted by subscripts. The time evolution of the polarity field $\mathbf{p} = (p_\mathrm{x}, p_\mathrm{y}, p_\mathrm{z})^\top$ is governed by Equation (3.1a). The

[1] Repeated index $\gamma$ means summation over the indices. Note that $\gamma$, the physical parameter should not to be confused with $_\gamma$ the coordinate

co-rotational Lagrangian derivative is defined as:

$$\frac{\mathrm{D}p_\alpha}{\mathrm{D}t} = \frac{\partial p_\alpha}{\partial t} + v_\gamma \partial_\gamma p_\alpha + \omega_{\alpha\beta} p_\beta, \tag{3.2}$$

where $\omega_{\alpha\beta} = \frac{1}{2}\left(\partial_\alpha v_\beta - \partial_\beta v_\alpha\right)$ is the vorticity tensor. $u_{\alpha\beta} = \frac{1}{2}\left(\partial_\alpha v_\beta + \partial_\beta v_\alpha\right)$ is the strain rate tensor, $\gamma$ is the rotational viscosity of the polarity field, $\nu$ is the coupling coefficient for mechanical stress and polarization that controls the flow-aligning ($|\nu| > 1$) or flow-tumbling ($|\nu| < 1$) nature of the active fluid. $\lambda$ is the coefficient coupling the polarity dynamics with the active chemical potential $\Delta\mu$. The molecular field $h_\alpha = -\delta F/\delta p_\alpha$ is the functional derivative of the Frank free energy density

$$F_{3D} = \frac{K_s}{2}(\nabla \cdot \mathbf{p})^2 + \frac{K_t}{2}(\mathbf{p} \cdot (\nabla \times \mathbf{p}))^2 + \frac{K_b}{2}(\mathbf{p} \times (\nabla \times \mathbf{p}))^2 - \frac{1}{2}h_\parallel^0 \|\mathbf{p}\|^2 \tag{3.3}$$

with the elastic constants $K_s$, $K_b$, and $K_t$ for splay, bend, and twist, respectively. The Lagrange multiplier $h_\parallel^0$ is used to enforce unit magnitude of the polarity. Equation (3.1b) is the force balance with the total stress tensor $\sigma_{\alpha\beta}^{(\mathrm{tot})}$ as the sum of symmetric, anti-symmetric, and Ericksen stresses. The pressure $\Pi$ is used as another Lagrange multiplier to enforce the incompressibility condition on the velocity field, $\nabla \cdot \mathbf{v} = 0$. Equation (3.1c) is the constitutive stress-strain relation with the symmetric stress $\sigma_{\alpha\beta}^{(\mathrm{s})}$, the fluid viscosity $\eta$, and the coefficient $\zeta$ coupling chemical activity $\Delta\mu$ to active stress. Restricting $\Delta\mu > 0$, the sign of $\zeta$ controls the contractile or extensile nature of the active fluid with $\zeta < 0$ leading to contractile stress and $\zeta > 0$ to extensile stress. The molecular field $h_\alpha$ can be decomposed into parallel $h_\parallel = p_\alpha h_\alpha$ and perpendicular $h_{\perp,\alpha} = \varepsilon_{\alpha\beta\gamma} p_\beta h_\gamma$ components in a local co-moving frame. This implies

$$h_x = h_\parallel p_x - h_{\perp,z} p_y + h_{\perp,y} p_z, \qquad h_{\perp,x} = p_y h_z - p_z h_y, \tag{3.4a}$$

$$h_y = h_{\perp,z} p_x + h_\parallel p_y - h_{\perp,x} p_z, \qquad h_{\perp,y} = p_z h_x - p_x h_z, \tag{3.4b}$$

$$h_z = -h_{\perp,y} p_x + h_{\perp,x} p_y + h_\parallel p_z, \qquad h_{\perp,z} = p_x h_y - p_y h_x. \tag{3.4c}$$

Since molecular fields that differ by a factor of $h_\parallel^0 p_\alpha$ are equivalent [144], $h_\parallel = h_\parallel^0$ becomes the Lagrange multiplier to enforce unit polarity magnitude, i.e., the vector field $\mathbf{p}$ indicates the direction of nematic order with $\|\mathbf{p}\| = 1$. Using Eq. (3.1a) and enforcing $p_\gamma \frac{Dp_\gamma}{Dt} = 0$, which is equivalent to constraining the Frank free energy to only allow unit polarity magnitude, we find the Lagrange multiplier:

$$h_\parallel^0 = h_\parallel = -\gamma\Big[\lambda\Delta\mu - \frac{2\nu}{p_\gamma p_\gamma}\Big(u_{\alpha\beta} p_\alpha p_\beta\Big)\Big]. \tag{3.5}$$

The total deviatoric stress tensor $\sigma_{\alpha\beta}^{(tot)}$ can be decomposed into its symmetric part $\sigma_{\alpha\beta}^{(s)}$, its antisymmetric part $\sigma_{\alpha\beta}^{(a)}$, and the Ericksen stress tensor $\sigma_{\alpha\beta}^{(e)}$. The symmetric stresses $\sigma_{\alpha\beta}^{(s)}$ are given by Eq. (3.1c), whereas $\sigma_{\alpha\beta}^{(a)}$ and $\sigma_{\alpha\beta}^{(e)}$ are defined as

$$\sigma^{(a)}_{\alpha\beta} = \frac{1}{2}(p_\alpha h_\beta - p_\beta h_\alpha)\,, \qquad \sigma^{(e)}_{\alpha\beta} = -\frac{\partial f}{\partial(\partial_\beta p_\gamma)}\partial_\alpha p_\gamma\,. \tag{3.6}$$

The pressure $\Pi$ acts as another Lagrange multiplier to enforce the incompressibility condition on the velocity field, $\partial_\gamma v_\gamma = 0$. Because of the Lagrange multiplier $h_\parallel$ entering the stress tensor, additional flow-field derivatives appear in the force-balance equation when computing the divergence of the total stresses $\partial_\beta \sigma^{(tot)}_{\alpha\beta}$. This leads to the final governing equations as given in Appendices B.1, B.2. Since those equations would be challenging to hand-code into a computer programming language, we leverage a C++ template expression system described in section 2.2 to directly translate the mathematical expressions into executable code.

## 3.3 Discretization-Corrected Particle Strength Exchange (DC-PSE)

DC-PSE is a numerical method for discretizing differential operators on irregular distributions of collocation points [77]. The method was originally derived as an improvement over the classic Particle Strength Exchange (PSE) [145] scheme, reducing its quadrature error on irregularly distributed collocation points, but mathematically amounts to a generalization of finite differences [77]. The PSE/DC-PSE class of collocation methods uses mollification with a symmetric smoothing kernel $\eta_\epsilon(\cdot)$ to approximate (sufficiently smooth) continuous functions $f(\boldsymbol{x}) \in \mathbb{R},\ \boldsymbol{x} \in \Omega \subseteq \mathbb{R}^d$,

$$f(\boldsymbol{x}_p) \approx f_\epsilon(\boldsymbol{x}_p) = \int_\Omega f(\boldsymbol{x})\,\eta_\epsilon(\boldsymbol{x}_p - \boldsymbol{x})\,\mathrm{d}\boldsymbol{x}, \tag{3.7}$$

where $f_\epsilon(\boldsymbol{x}_p)$ is a regularized approximation of the function $f$ at location $\boldsymbol{x}_p \in \Omega$ of collocation point $p$. The scalar $\epsilon$ is the smoothing length (or the kernel width) of the mollification. Linear differential operators in $\mathbb{R}^d$,

$$\boldsymbol{D}^{\boldsymbol{\alpha}} = \frac{\partial^{|\boldsymbol{\alpha}|}}{\partial x_1^{\alpha_1}\partial x_2^{\alpha_2}\cdots\partial x_d^{\alpha_d}}\,, \tag{3.8}$$

defined by the multi-index $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d) \in \mathbb{Z}^d$ with $|\boldsymbol{\alpha}| = \sum_{i=1}^d \alpha_i$ are approximated by Taylor series expansion to find a discrete operator

$$\boldsymbol{Q}^{\boldsymbol{\alpha}} f(\boldsymbol{x}_p) = \boldsymbol{D}^{\boldsymbol{\alpha}} f(\boldsymbol{x}_p) + O\big(h(\boldsymbol{x}_p)^r\big) \tag{3.9}$$

at collocation point $\boldsymbol{x}_p$. The order of approximation $r$ depends on the kernel $\eta_\epsilon$ used in Eq. (3.7), and $h(\boldsymbol{x}_p)$ is the average distance between collocation point $p$ and its neighbors within the kernel support. The Taylor expansion yields integral constraints (also known

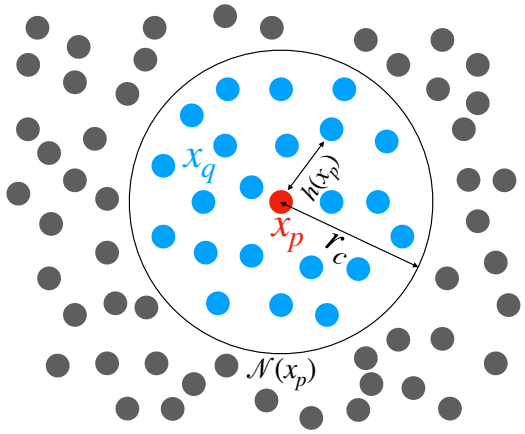



Figure 3.1: Illustration of the DC-PSE method. The collocation points $\boldsymbol{x}_q$ (blue) within the symmetric operator support $\mathcal{N}(\boldsymbol{x}_p)$ of radius $r_c$ around the center point $\boldsymbol{x}_p$ (red) are used to approximate the differential operator at $\boldsymbol{x}_p$. The average distance between points in the operator support, $h(\boldsymbol{x}_p)$ defines the accuracy of the approximation.

as *continuous moment conditions*), which the kernel $\eta_\epsilon$ needs to fulfill in order to reach a certain order $r$ [145].

DC-PSE uses different kernels $\eta_\epsilon^p(\cdot,\cdot)$ for different collocation points $p$ and directly acts on a given quadrature of Eq. (3.7) with collocation points $\boldsymbol{x}_q \in \Omega$, resulting in the discrete operator:

$$\boldsymbol{Q}_h^{\boldsymbol{\alpha}} f(\boldsymbol{x}_p) = \frac{1}{\epsilon(\boldsymbol{x}_p)^{|\boldsymbol{\alpha}|}} \sum_{\boldsymbol{x}_q \in \mathcal{N}(\boldsymbol{x}_p)} \big(f(\boldsymbol{x}_q) \pm f(\boldsymbol{x}_p)\big)\, \eta_\epsilon^p(\boldsymbol{x}_p, \boldsymbol{x}_q), \tag{3.10}$$

where $\mathcal{N}(\boldsymbol{x}_p)$ are all collocation points in the neighborhood (of a certain radius $r_c$ defined by the kernel width) around point $\boldsymbol{x}_p$, as illustrated in Fig. 3.6a. The positive sign in the parenthesis is used for odd $|\boldsymbol{\alpha}|$, the negative sign for even $|\boldsymbol{\alpha}|$. This renders the operator conservative on symmetric collocation point distributions, i.e., when $\eta_\epsilon^p(\boldsymbol{x}_p, \boldsymbol{x}_q) = \eta_\epsilon^q(\boldsymbol{x}_q, \boldsymbol{x}_p)$. In DC-PSE, the kernels $\eta_\epsilon^p$ are thus not determined from continuous moment conditions, as in PSE, but directly from the *discrete moment conditions* that result from substituting Eq. (3.10) into the quadrature of Eq. (3.7) [77] for a given set of $\{\boldsymbol{x}_q\}_{q=1}^N$. This adapts the kernels to the specific distribution of collocation points (hence the name "discretization-corrected") and avoids the quadrature error of PSE [145], leading to a scheme that is consistent with order $r$ on (almost[2]) arbitrary collocation point sets. This means that at each collocation point, a potentially different kernel is used for the same differential operator if the neighboring collocation points within the kernel support are distributed differently.

[2]The collocation point distribution must not be degenerate in the sense that the Vandermonde matrix of the kernel system must have full rank [78]. A trivial example: placing all points along a line and then asking for an approximation of the derivative in the perpendicular direction cannot work.

Evaluating such a kernel at the locations of the collocation points yields a generalized finite-difference stencil, which reduces to the classic compact finite differences on regular grid arrangements of points [77].

DC-PSE kernels are determined at runtime by solving a small system of linear equations for each collocation point, resulting from the discrete moment conditions in its kernel neighborhood. For this, one can choose the function space such that the kernels are compact and symmetric. A frequent choice are polynomials windowed by truncated exponentials [79]

$$\eta_\epsilon^p(\boldsymbol{x}_p, \boldsymbol{x}_q) = \eta_\epsilon^p\left(\frac{\boldsymbol{x}_p - \boldsymbol{x}_q}{\epsilon(\boldsymbol{x}_p)}\right) := \left(\sum_{|\gamma|=\beta_{\min}}^{|\boldsymbol{\alpha}|+r-1} a_\gamma(\boldsymbol{x}_p)\left(\frac{\boldsymbol{x}_p - \boldsymbol{x}_q}{\epsilon(\boldsymbol{x}_p)}\right)^\gamma\right) \mathrm{e}^{-\left|\frac{\boldsymbol{x}_p - \boldsymbol{x}_q}{\epsilon(\boldsymbol{x}_p)}\right|^2} \tag{3.11}$$

of finite radius $r_c$. The polynomial coefficients $a_\gamma$ are determined for a given $\boldsymbol{\alpha}$ and given collocation points $\boldsymbol{x}_q \in \mathcal{N}(\boldsymbol{x}_p)$, such that the following discrete moment conditions are satisfied [77]:

$$Z_h^{\boldsymbol{\beta}} = \begin{cases} (-1)^{|\boldsymbol{\alpha}|}\boldsymbol{\alpha}!, & \boldsymbol{\beta} = \boldsymbol{\alpha} \\ 0, & \boldsymbol{\beta} \neq \boldsymbol{\alpha}, \quad \beta_{\min} \leqslant |\boldsymbol{\beta}| \leqslant |\boldsymbol{\alpha}| + r - 1, \ \beta_{\min} = \begin{cases} 0, & |\boldsymbol{\alpha}| \text{ odd} \\ 1, & |\boldsymbol{\alpha}| \text{ even} \end{cases} \\ < \infty, & |\boldsymbol{\beta}| = |\boldsymbol{\alpha}| + r \end{cases} \tag{3.12}$$

where

$$Z_h^{\boldsymbol{\beta}}(\boldsymbol{x}) = \frac{1}{\epsilon(\boldsymbol{x}_p)^d} \sum_{\boldsymbol{x}_q \in \mathcal{N}(\boldsymbol{x}_p)} \frac{(\boldsymbol{x}_p - \boldsymbol{x}_q)^{\boldsymbol{\beta}}}{\epsilon(\boldsymbol{x}_p)^{|\boldsymbol{\beta}|}} \eta_\epsilon^p\left(\frac{\boldsymbol{x}_p - \boldsymbol{x}_q}{\epsilon(\boldsymbol{x}_p)}\right) \tag{3.13}$$

is the discrete moment of order $\boldsymbol{\beta}$ of the kernel $\eta_\epsilon^{\boldsymbol{\alpha}}$, and $\beta_{\min}$ is the parity of $|\boldsymbol{\alpha}|$, because the zeroth moment $Z_h^{\mathbf{0}}$ vanishes for even operators. Under these conditions, DC-PSE is consistent with order $r$ as long as

$$\frac{h(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)} \in O(1), \tag{3.14}$$

i.e., the kernel width $\epsilon$ scales proportionally with the average inter-point distance $h$ around $\boldsymbol{x}_p$ [77].

## 3.4 Mesh-free algorithm for solving the active Ericksen-Leslie model

We start with describing the force-balance solver, because knowing the velocity field is prerequisite for evolving the polarity in time. We then outline the computation of the polarity time-evolution and discuss advantages and disadvantage of Eulerian and Lagrangian

frames of reference. Finally, we summarize the proposed hybrid particle-mesh method with normalization correction for robust simulation of 3D active fluids. All computer codes are implemented in C++ using the open-source scalable scientific computing framework OpenFPM [142].

### 3.4.1 Force-balance solver

**Algorithm 3.1** Incompressible force-balance solver for a given polarity field $\mathbf{p}$.

**Input:**

1. $\epsilon_v$: Numerical relative tolerance for the incompressible flow solver.
2. $n_{\max}$: Maximum number of pressure-correction iterations.
3. $\mathbf{v}^0, \Pi^0$: Initial guess for velocity and pressure

**Output:** $\mathbf{v}$ : Incompressible flow velocity satisfying Eq. (3.1b) and the boundary conditions.

$n = 0$, $\epsilon = \infty$
**while** $\|\mathbf{v}^n - \mathbf{v}^{n-1}\|_2 < \epsilon_v$ *and* $n <= n_{\max}$ **do**
    Solve Stokes equations for $\mathbf{v}^n$ using $\Pi^{n-1}$
    Correct pressure: $\Pi^n = \Pi^{n-1} - \partial_k v_k^n$
    **if** $\|\mathbf{v}^n - \mathbf{v}^{n-1}\|_2 > \epsilon$ **then**
        Stop with error: "Requested tolerance is unachievable for the chosen resolution".
    $\epsilon = \|\mathbf{v}^n - \mathbf{v}^{n-1}\|_2$
    $\mathbf{v}^{n-1} = \mathbf{v}^n$
    $n = n + 1$

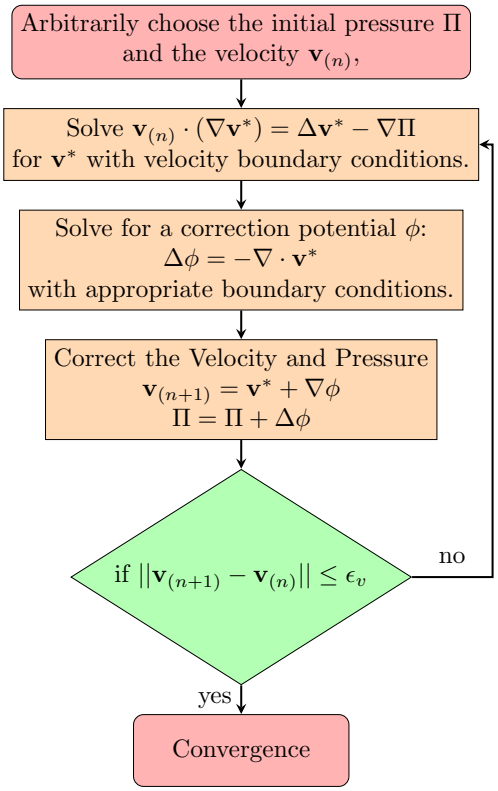



Figure 3.2: Flow diagram of the pressure-correction algorithm for steady-state incompressible Navier-Stokes simulations with user-provided numerical tolerance $\epsilon_v$.

Given the polarity field $\mathbf{p}$ for any point in time $t$, we determine the induced velocity field $\mathbf{v}$ by discretizing Eqs. (B.12a)–(B.12c) using DC-PSE differential operators [141]. Incompressibility is imposed using a pressure-correction scheme as previously described in 2D [135, 137] and successfully used with DC-PSE operators [146]. We extend this algorithm to 3D. The solver can be used on irregular distributions of collocation points, which we

therefore refer to as *particles*. This renders the algorithm suitable for simulations in both Eulerian and Lagrangian frames of reference, where particles are stationary or move with the velocity field of the flow, respectively. The resulting linear system of equations is assembled using the OpenFPM template expression system [143] and is solved with the KSPGMRES solver from the PETSc library [98]. The pressure-correction algorithm iteratively improves the estimate of the flow velocity to a desired numerical tolerance $\epsilon_v$. The tolerance $\epsilon_v$ is bounded from below by the approximation error of the spatial derivatives, which converges with increasing resolution. We set it to $10^{-2}$ for $h = 0.2$ and $10^{-3}$ for $h = 0.1$. We describe the full pressure-correction solver in Algorithm 3.1 and Fig. 3.2. Once the velocity field is obtained, we perform the time integration step for evolution of the polarity field.

### 3.4.2 Time evolution of polarization field

Once the velocity field has been computed, we perform the time-integration step for the polarity field. Equation (3.1a) can be integrated in time in either an Eulerian or Lagrangian frame of reference until a desired final time $t_f$. In an Eulerian frame of reference, the advection operator in Eq. (3.2) is be computed using DC-PSE spatial operators. To achieve numerical stability, however, predictor-corrector time integration is necessary, as the operators are centered in space and not upwind. Hence, for the Eulerian frame of reference, we use the second-order Adams-Bashforth-Moulton predictor-corrector scheme for time integration, as implemented in OpenFPM. For Eulerian simulations, the time step $\delta t$ is limited by the CFL condition $\frac{u_x \delta t}{h_x} + \frac{u_y \delta t}{h_y} + \frac{u_z \delta t}{h_z} \leq 1$ for numerical stability [147]. This severely limits the time-step size. The issue is exacerbated at higher activity $|\zeta|$, leading to stronger active stresses causing faster flows. However, when computing flow steady-states, and for imposing boundary conditions in complex domain shapes, the Eulerian frame of reference still provides better stability than the Lagrangian approach. Steady states are detected with a user-defined tolerance of $\epsilon_{\text{steady}}$, checked against the maximal rate of change of polarity in the entire domain.

In the Lagrangian frame of reference, the advection Eq. (3.1a) decomposes into two equations,

$$\frac{\mathrm{d}x_\alpha}{\mathrm{d}t} = v_\alpha \,, \tag{3.15a}$$

$$\frac{\partial p_\alpha}{\partial t} = \frac{h_\alpha}{\gamma} - \nu u_{\alpha\beta} p_\beta + \lambda \Delta \mu p_\alpha + \omega_{\alpha\beta} p_\beta \,, \tag{3.15b}$$

allowing for the less stringent Lagrangian CFL condition $\delta t \|\nabla \mathbf{v}\| \leq 1$, imposing that particle trajectories never cross flow streamlines [148]. Moving the particles changes their positions $x_\alpha$, which requires DC-PSE kernels to be recomputed at each simulation time step. Time integration in the Lagrangian case is also done using the second-order Adams-Bashforth-Moulton predictor-corrector scheme. Alternatively, Runge-Kutta 4 can be used in this case. This adds additional computational cost, but is amortized in most cases by the increased

stability permitting larger time steps [141]. Detecting non-equilibrium steady states with non-zero flow, however, is not easily done in the Lagrangian frame of reference. It is only possible if the rate of change of the polarity can be equated to an approximation of the advection operator. Computing the advection operator, however, introduces additional numerical error and lowers the tolerance with which the steady state can be detected. Nevertheless, the superior time-stepping stability of the Lagrangian method makes it the preferred choice for simulating unsteady flows in simple Cartesian geometries.

### 3.4.3 Remeshing

Another issue in Lagrangian simulations is that particles may eventually evolve to a distorted distribution at long simulation times of steady flows. Then, *remeshing* becomes necessary [148], whereby the particle positions are reset to a regular Cartesian lattice and the field quantities are "interpolated" from the old to the new set of particles. Multiple methods are available for moment-conserving particle-to-mesh remeshing[3]. Here, we use the lambda kernels $\Lambda_{m,n}$, which exactly conserve the first $m$ moments of the field and are of smoothness class $\mathcal{C}^n$ [148]. However, using such conservation-enforcing kernels only leads to higher-order (>1) convergent results if the zeroth moment of the remeshed field is actually conserved over time [149]. In the present solver, we only remesh the polarity field, since the velocity field is recomputed from the force balance at every time step. Since the polarity field is constrained to unit magnitude, its zeroth moment is constant. This justifies the use of moment-conserving remeshing kernels, which converge with the desired order in this case. Specifically, we use the $\Lambda_{2,1}$ kernel [148], which is equivalent to the classic $M_4'$ kernel [150], for remeshing Lagrangian particles:

$$\Lambda_{2,1}(d) = M_4'(d) = \begin{cases} 1 - \frac{1}{2}(5d^2 - 3d^3) & 0 \leq d < 1, \\ \frac{1}{2}(2-d)^2(1-d) & 1 \leq d \leq 2, \\ 0 & d > 2, \end{cases} \tag{3.16}$$

where $d$ is the distance of a particle to a grid node, measured in units of the grid spacing $h$. The algorithm loops through all particles and, for each particle, over all mesh nodes in the support of the $\Lambda_{2,1}$ kernel around it. For each particle-node pair, the above formula is used to compute the fraction of the particle's field value that is attributed to that mesh node. The mesh nodes accumulate (sum) the contributions they receive from all the particles. We remesh the polarity field after every advection time step. To ensure full particle support for the interpolation kernel close to boundaries, we use the method of images [151]. This method mirrors all particles within a distance of $3h$ from a boundary along the boundary normal. Specifically, if a particle $p$ inside the domain carries a polarity of $\mathbf{p}_p$ and is distance $\ell$ away from the boundary, a mirror particle will be created at distance $-\ell$ with a polarity

[3]Although these methods are commonly referred to as "interpolation" in the literature, they are not actually interpolating, as they do in general not maintain the exact field values at the original data points.

of $\mathbf{p}_{\text{mirror}} = 2\mathbf{p}_b - \mathbf{p}_p$, where $\mathbf{p}_b$ is the polarity boundary condition. The algorithm also takes care of renormalizing the polarity magnitude to compensate for errors accumulated during polarity time evolution. This is necessary, as these errors would otherwise amplify to larger numerical errors for long-time simulations. Hence, we project the polarity onto the unit sphere if its magnitude deviates from 1 by more than a user-defined tolerance $\epsilon_p$. The pseudo-code for the complete Lagrangian advection with remeshing is given in Algorithm 3.2. It is a hybrid particle-mesh method. During the time evolution of polarity,

**Algorithm 3.2** Lagrangian advection with remeshing

**Input:**

1. $S_0$: Particle set with polarity $\mathbf{p}$ and velocity $\mathbf{v}$ stored on the particles located on a mesh.

2. $\epsilon_p$: Numerical tolerance for deviation in polarity magnitude.

**Output:** Polarity field $\mathbf{p}$ on $S_0$ after advection with velocity $v$.

1: Create a particle distribution $S_1$ by advecting particles from $S_0$.
2: Create mirror particles outside the domain boundaries as required for the method of images to impose boundary conditions [151].
3: Interpolate polarization $\mathbf{p}$ from $S_1$ to $S_0$.
4: **for** each particle **do**
5: **if** $(1 - \|\mathbf{p}\|_2) > \epsilon_p$ **then**
6: $\mathbf{p} = \frac{\mathbf{p}}{\|\mathbf{p}\|_2}$

numerical error in the magnitude of polarization is accumulated and can lead to very large numerical error for long time simulations. Hence, we project the polarity on the unit sphere if the magnitude deviates by a tolerance $\epsilon_p$ from unity as shown in line 5-6 of algorithm 3.2.

The theoretically expected order of convergence of the method presented above depends on the spatial discretization operators and the time-integration scheme used. Here, we discretize all spatial differential operators using DC-PSE with consistency of $O(N^{-2})$, where $N$ is the total number of particles used in the simulation. The $\Lambda_{2,1}$ remeshing scheme converges with third order $O(N^{-3})$ and does therefore not limit the convergence rate. Time is integrated using the 2-step Adams-Bashforth-Moulton predictor-corrector scheme, which has a global convergence order of $O(N_t^{-2})$, where $N_t$ is the total number of time steps in a simulation. We therefore expect the method to overall be second-order accurate in both space and time.

### 3.4.4 Validation - 3D active incompressible film

We validate the numerical convergence of the present solver (and its software implementation) in a case with $\nu = 0$ for which an asymptotic analytical solution can be derived in 3D.

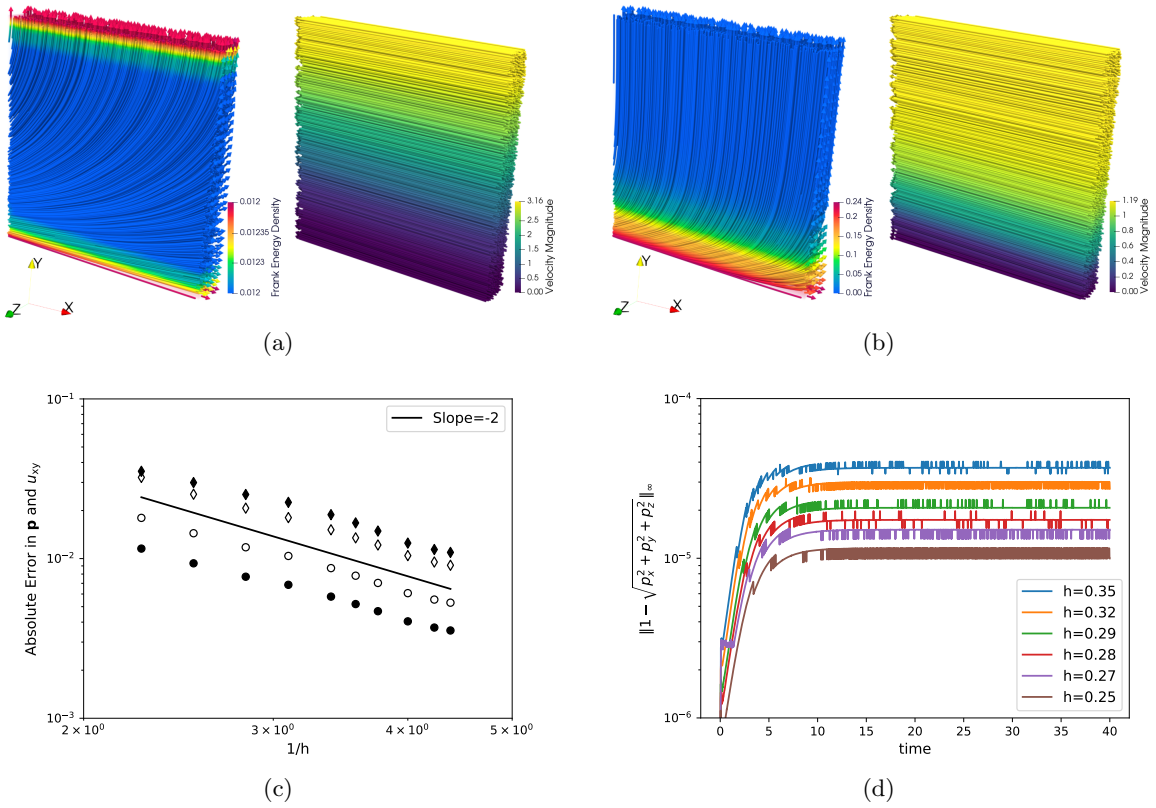



Figure 3.3: Convergence of the presented 3D active polar fluid solver for the thick-film problem with the analytical solution given in Eq. (3.18). **(a)** Streamlines of initial ($t = 0$) polarity (left) and velocity (right) fields with arrow heads indicating direction and color indicating the Frank energy density and velocity magnitude, respectively. **(b)** Streamlines at steady state ($t = 40$) for the polarity (left) and velocity (right) fields. **(c)** Convergence of the absolute numerical errors at $t = 40$ for the polarity $\mathbf{p}$ and strain rate $u_{xy}$ fields. The $L_2$ norm of the errors across all particles is plotted as circles ($\circ$), the $L_\infty$ norms as diamonds ($\Diamond$). Open symbols represent errors in the polarity field $\mathbf{p}$, filled symbols in the strain rate field $u_{xy}$. The theoretically expected convergence slope is indicated by the solid line. **(d)** Evolution of the $L_\infty$ error in the numerically computed polarity magnitude $\|\mathbf{p}\|_2$ over time for different resolutions $h$ demonstrating convergence of the Lagrange multiplier $h_\parallel$.

Consider a thick film that is infinitely long along the $x$ and the $z$ directions and has thickness $L$ in the $y$ direction. The surface of the film at $y = L$ is stress-free ($\sigma_{xy}^{(\mathrm{tot})}(x, L, z, t) = 0$), has no slip in $z$, ($v_z(x, L, z, t) = 0$), and is impenetrable ($v_y(x, L, z, t) = 0$). The surface at $y = 0$, is also impenetrable ($v_y(x, 0, z, t) = 0$) and has no slip in $x$ and $z$ ($v_x(x, 0, z, t) =$

$v_z(x,0,z,t) = 0$). The polarity is anchored at both surfaces as $(p_x,p_y,p_z)(x,L,z,t) = (0,1,0)$ and $(p_x,p_y,p_z)(x,0,z,t) = (0,1,0)$. Under these conditions, $v_y = 0$ everywhere due to incompressibility and translational invariance in $x$ and $z$. For the initial condition

$$\mathbf{p}(x,y,z,0) = \left(\cos\left(\frac{\pi y}{2L}\right), \sin\left(\frac{\pi y}{2L}\right), 0\right), \tag{3.17}$$

we can derive the asymptotic steady-state solution

$$\mathbf{p}(x,y,z,t\to\infty) = \big(\cos\theta(y), \sin\theta(y), 0\big), \tag{3.18a}$$

$$u_{xy}(x,y,z,t\to\infty) = u_{yx}(x,y,z,t\to\infty) = \frac{K}{\gamma[1+\nu\cos\theta(y)]}\frac{\mathrm{d}^2\theta(y)}{\mathrm{d}y^2}. \tag{3.18b}$$

For $\nu = 0$, the hydrodynamic equations reduce to

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}y^2} = \frac{\Delta\mu\gamma\zeta}{K(4\eta+\gamma)}\sin 2\theta \tag{3.19}$$

with

$$\theta(y) = \mathrm{am}\left(\sqrt{c_1-a}\,(c_2+y), -\frac{2a}{c_1-a}\right), \tag{3.20}$$

where $\mathrm{am}(u,k)$ is the Jacobi amplitude function. This solution in 3D is invariant along $z$ and is therefore an extension of a known 2D problem [151].

We numerically solve this problem for $\eta = 1$, $\gamma = 1$, $\zeta = -1$, $\lambda = 1$, $K_s = K_b = K_t = 1$, $\Delta\mu = -1$, and $L = 10$ in the Lagrangian frame of reference with remeshing. For this set of parameters, the analytical solution has $\theta(y) = \mathrm{am}\left(\sqrt{0.2(1+y)}, -0.5\right)$. We use the proposed hybrid particle-mesh method with $\epsilon_v = 10^{-5}$, $\epsilon_p = 10^{-16}$, $n_{\max} = 30$, and $\epsilon_{\text{steady}} = 10^{-9}$ to perform the simulation in a box-shaped domain of size $(10,10,5h)$ discretized by a uniform Cartesian mesh of resolution $h$. The domain has periodic boundary conditions in $x$ and $z$ to model the infinite extent of the fluid film in these directions. In the $y$ direction, we use the boundary conditions as specified by the problem.

Refining $h$, we observe second-order convergence to the analytical solution at $t = 40$, as theoretically expected. In order to satisfy the CFL condition, the time-step size is refined along with the grid resolution according to the relation $N_x = \sqrt{0.64N_t + 1}$, where $N_x, N_t$ are the number of particles in $x$ direction and number of time steps to reach $t = 40$, respectively. The initial polarity and velocity fields at $t = 0$ are visualized in Fig. 3.3a. The steady-state numerical solution at $t = 40$ is visualized in Fig. 3.3b. We plot the $L_2(\circ,\bullet)$ and $L_\infty(\lozenge,\blacklozenge)$ errors in both the polarity field $\mathbf{p}$ and the non-zero components of the strain rate $u_{xy}$ for increasing $N_x$ ($h = 1/N_x$) in Fig. 3.3c. We further verify in Fig. 3.3d that the error in the polarity magnitude $\|\mathbf{p}\|$ plateaus over time and converges for decreasing

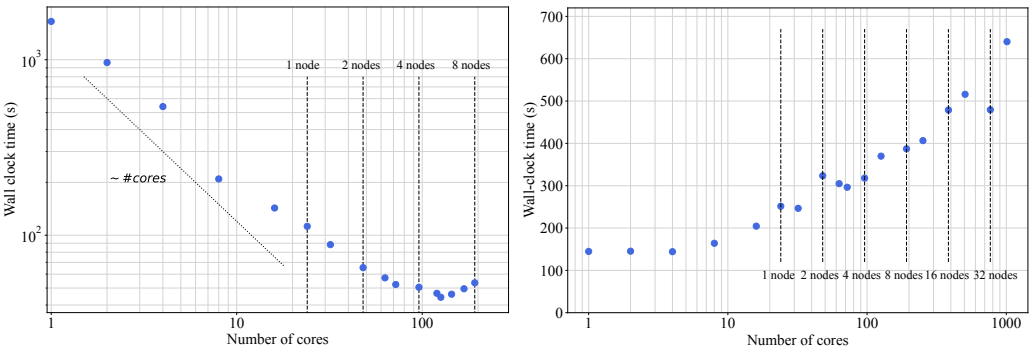



Figure 3.4: Scaling behaviour of the hybrid particle-mesh algorithm. (a) Strong scaling measured by the wall-clock time depending on the number of cores. The dotted line represents the linear slope. (b) Weak scaling measured by the efficiency. After a stronger initial decrease, the efficiency plateaus for higher particle and core numbers.

$h$, confirming numerical convergence of the Lagrange multiplier $h_{\|}$. Together with earlier convergence results of the DC-PSE pressure-correction scheme for steady-state Stokes flow in a 3D ball with irregular particle distributions [143], this validates the correctness of the present method also in the active polar case.

The convergence on a irregular particle distribution using DC-PSE and the pressure correction scheme was shown for an incompressible steady-state Stokes flow in a 3D ball in [143] which underlines the power of DC-PSE for complex geometries.

### 3.4.5 Computational cost and scalability

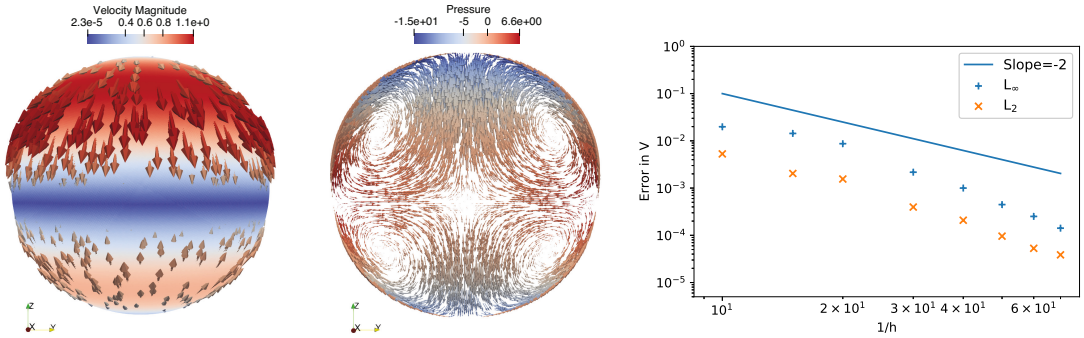
Next, we look at the computational performance of the solver. The KSPGMRES algorithm [152] implemented in the PETSc library uses the Arnoldi iterations to find the orthonormal basis of the Krylov subspace. In each Arnoldi iteration, a new basis vector is found and orthonormalized using the Gram-Schmidt process. This requires storing the vectors from all $k$ previous iterations. The number of multiplications over $k$ Arnoldi iterations thus is $O(\frac{1}{2}k^2N)$, where $k$ in the worst case is $N$, and the storage requirement is $O(Nk)$. We prevent the explosion of the cost with $k^2$ by restarting the solver every $m = 5000$ iterations. This restarted version requires $O((m+3+1/m)N+Z)$ multiplications, where $Z$ is the number of nonzero elements in the sparse system matrix. The memory requirement reduces to $O((m + 2)N)$. For each iteration of the pressure-correction algorithm, the KSPGMRES solver is run to convergence. This is repeated at most $n_{\max}$ times.

The cost for the 2-step Adams-Bashforth-Moulton time-integration scheme is $O(N)$.

Likewise, remeshing has a computational cost of $O(N)$, as each particle assigns to a constant number of neighboring grid nodes. In the present Eulerian simulation, the particle positions do not change over time. Therefore, the DC-PSE kernels only need to be calculated once at the beginning of the simulation. Evaluating the kernels is an $O(N)$ operation. In summary, the overall computational cost of the simulation in an Eulerian frame of reference is dominated by KSPGMRES, as it needs to solve a linear system of equations for each pressure-correction iteration, the cost of which depends on the condition number of the system. In a Lagrangian frame of reference, the DC-PSE kernels need to be recomputed at every time step, incurring an additional computational cost of $O(N)$, albeit with a large pre-factor. This cost then dominates the one from the linear system solver.

The software implementation of the presented method can leverage shared- and distributed-memory parallel computers to address the high computational cost of 3D simulations. We therefore check how the runtime of the simulation from Sec. 3.4.4 scales when distributing it across an increasing number of CPU cores. For each configuration, ten time steps are performed on Intel Xeon E5-2680v3 CPUs at 2.50 GHz with 24 cores on each compute node. Individual compute nodes are connected by a 4-lane FDR InfiniBand network (at 14 Gb/s per lane) with a latency of 0.7 µs for message passing using the OpenMPI library. Each experiment is repeated 10 times.

For a fixed number of particles ($N_x = 64$, $N_y = 65$, $N_z = 5$), we increase the number of CPU cores form 1 to 192, hence measuring the *strong* scaling of the algorithm (Fig. 3.4a). The time measurements show an almost linear scaling within one compute node. Scaling beyond one node, communicating data via the interconnect network of the machine, the scaling slows down only slightly up until 128 cores. Using more than 128 cores, runtimes start to increase, as this problem is too small to be distributed over more than 128 cores. We next measure the *weak* scaling of the simulation where the number of particles increases proportionally to the number of CPU cores. For simplicity, we perform this test in a cubic computational domain and increase the number of particles from $N = 16 \times 17 \times 16$ on 1 core to $N = 164 \times 165 \times 164$ (one additional particle is required in $y$ to impose the boundary conditions) on 1012 cores (Fig. 3.4b). We observe perfect scaling in one node up to 4 cores. Using more cores per node, the memory bandwidth becomes the bottleneck, such that the speedup on 24 cores(1 full node) is only about 14-fold. Scaling to more nodes, the communication overhead of the linear system solver eventually becomes limiting, as it requires global all-to-all communication with a volume that grows with the total problem size. Taken together, these results demonstrate that the parallel OpenFPM implementation of the present code can reduce the wall-clock time of the simulations by more than one order of magnitude (strong scaling) and allows performing large simulations (weak scaling). This enables using the code for real-world problems.




(a) (b) (c)

Figure 3.5: Visualization and convergence for Stokes flow inside a three-dimensional unit ball. **(a)** Visualization of the flow velocity (arrows colored by magnitude) in comparison with the analytical solution for mode $l = 2$ (magnitude: solid background color). **(b)** Velocity (arrows) and pressure (color) visualized in the $y - z$ plane cut through the ball's center. **(c)** Convergence plot showing the $L_2$ and $L_\infty$ norms of the absolute error in the velocity field computed against the analytical solution for $l = 2$ for different average inter-particle spacing $h$. The solid line shows the theoretically expected scaling.

## 3.5 Validation of Stokes flow in 3D ball

We test the robustness and convergence of the Stokes flow solver in domains with curved boundaries as this is the main step required for solving active hydrodynamics in complex geometries. This step is critical, because although computing derivatives in a complex geometry with the DC-PSE meshless method is straightforward (as it is equivalent to generalized Finite Differences [127]), It is unclear that the resulting implicit linear systems would solve the global incompressibility problem accurately. To test this, we consider Stokes flow in a three-dimensional ball to demonstrate the robustness of our DC-PSE pressure correction scheme for consistent discretization of differential operators in curved domains. Although this example does not concern active flows, it is chosen because it allows us to test the numerical method with an analytically known solution. This example also shows the versatility expression system for PDEs in non-Cartesian domains without rewriting the PDEs themselves from the Lid driven cavity example in chapter 2. We validate this case by showing convergence to the analytical solution.

We numerically solve the incompressible steady-state Stokes flow equations

$$\Delta \mathbf{v} = \nabla \Pi, \qquad \mathbf{v} \in \Omega \backslash \partial\Omega \tag{3.21a}$$

$$\nabla \cdot \mathbf{v} = 0 \tag{3.21b}$$

in the closed unit ball $\Omega = \overline{B_1}(0) \subset \mathbb{R}^3$ centered at the origin. We solve these equations for

the velocity $\mathbf{v}$ and the pressure $\Pi$ by imposing velocity boundary conditions on the surface of the ball, given by the vector spherical harmonics with unit amplitude for the mode $l = 2$, $m = 0$. The analytical solution for this problem is then known [153] as:

$$\mathbf{v} = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} u^r_{lm}(r)\mathbf{Y}^{(lm)} + u^{(1)}_{lm}(r)\mathbf{\Psi}^{(lm)} + u^{(2)}_{lm}(r)\mathbf{\Phi}^{(lm)} \tag{3.22a}$$

$$\Pi = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} p_{lm}(r) Y_{lm}, \tag{3.22b}$$

where $\mathbf{Y}^{(lm)}$, $\mathbf{\Psi}^{(lm)}$, and $\mathbf{\Phi}^{(lm)}$ are the vector spherical harmonics, $Y_{lm}$ is the scalar spherical harmonic, and $u^r_{lm}(r)$, $u^{(1)}_{lm}(r)$, $u^{(2)}_{lm}(r)$, and $p_{lm}(r)$ are coefficients determined from the velocity boundary condition.

We use DC-PSE to discretize the differential operators in space and to build the system matrix. We then use the KSPGMRES solver from PETSc [98] as encapsulated by our expression system to numerically solve the resulting linear system of equations with pressure correction (see Section 2.6.1) to impose the incompressibility condition. This validates the mesh-free pressure correction scheme using DC-PSE also in a simulation domain with curved boundary.

Implementing this simulation using the present C++ expression system as described in the previous chapter, the scalable OpenFPM code is 180 lines long, which includes the initialization of the boundary conditions from numerically computed vector spherical harmonics. We validate the simulation by comparing against the analytical solution for different modes $l$ and $m$. We observe that for the mode $l = 1$, the error is limited by the tolerance $\epsilon$ of the pressure-correction iterations (see Fig. 3.2). Hence, we present the convergence of the numerical method for $l = 2$ in Fig. 3.5.

## 3.6 Surface DC-PSE

Partial Differential Equations (PDEs) on curved surfaces and differentiable manifolds are an important tool in understanding and studying physical phenomena such as surface flows [68, 154] or active morphogenesis [62]. Analytically solving intrinsic PDEs in curved surfaces, however, quickly becomes impossible for nonlinear PDEs or for surfaces that do not possess a global parameterization. Therefore, numerical methods for solving intrinsic PDEs on curved surfaces are important, and a wide variety of both embedded and embedding-free schemes have been developed to consistently discretize intrinsic differential operators over scalar fields on surfaces.

### 3.6.1 Related works

Embedding-free methods require a (at least local) parametrization of the surface in order to discretize the differential operators via coordinate charts or a local basis of the manifold [155]. This includes methods based on moving frames [156], a concept originally developed in continuous group theory, where the surface geometry is locally represented by orthonormal bases. The concept of moving frames has also been combined with discontinuous Galerkin discretization, e.g., to solve shallow-water equations on arbitrary rotating surfaces [157]. Other embedding-free Finite Element Methods (FEM) include intrinsic Surface FEM (ISFEM), which discretizes differential operators on a triangulation of the surface [158, 159], and methods based on Discrete Exterior Calculus (DEC) [160].

Embedding methods discretize the surface problem in the embedding space and use projections to restrict the differential operators computed in the embedding space to the surface. This includes methods that use explicit tracer points to represent the surface, but interpolate to an embedding mesh to evaluate differential operators [161], diffuse-interface methods based on phase-field representations of the surface [69], embedding FEM such as TraceFEM [162], narrow-band level-set methods based on orthogonal extension of the surface quantities [163,164], level-set methods based on the closest-point transform [165,166], and volume-of-fluid methods for surface PDE problems [167].

While each of these methods has its specific strengths, embedding methods usually generalize better to complex-shaped or arbitrary surfaces [165]. However, they tend to have higher computational cost, because computations are done in the higher-dimensional embedding space and additional extension (for level sets), right-hand-side evaluation (for phase fields), or interpolation (for closest-point transforms) steps are required, albeit specific optimizations are available, e.g., for level sets [168]. Also, embedding-free methods are generally more accurate because they avoid the interpolation and projection errors arising when the discretization of the embedding space does not trace the surface exactly, but they tend to be more difficult to implement and harder to generalize to complex-shaped surfaces.

In this section, we present a mesh-free collocation method for PDEs on smooth curved surfaces with non-intersecting tubular neighborhood (tubular network). The method combines elements from embedding and embedding-free approaches. The method is *algorithmically* embedding-free in the sense that surface quantities are represented on tracer points that are contained in the surface. This also discretizes and represents the surface itself. But the method is *mathematically* related to embedding approaches, since the stencils used to approximate differential operators at the surface points are computed in the embedding space by a reduction operation along the local normal vector, which needs to be known. Intuitively, this projects the discrete operators, rather than projecting the flux vectors as typically done in embedding methods. The resulting method therefore shares properties with moving frame approaches, such as the low dimensionality (and hence low computational cost) and the mesh-free character [156]. It combines these with properties of embedding methods, such as their flexibility in generalizing to complex surfaces [165], and

their ability to compute extrinsic differential-geometric quantities.

Our method is based on the Discretization-Corrected Particle Strength Exchange (DC-PSE) collocation scheme, which generalizes finite differences to arbitrary point clouds. Given the local surface normal $\boldsymbol{n}$, we derive intrinsic discrete operators by first creating an embedding narrow band and placing collocation points along the normal from each surface point. We then determine the regular DC-PSE operator kernels in the embedding space. These kernels are then reduced under the condition of normal extension $\nabla f \cdot \boldsymbol{n} = 0$ for any (sufficiently) differentiable scalar field $f(\boldsymbol{x}) \in \mathbb{R}$ to derive intrinsic kernels at the surface points $\boldsymbol{x}$. This is possible due to the kernel nature of DC-PSE, and it preserves the information from the embedding space in a scheme that only requires computation over surface points.

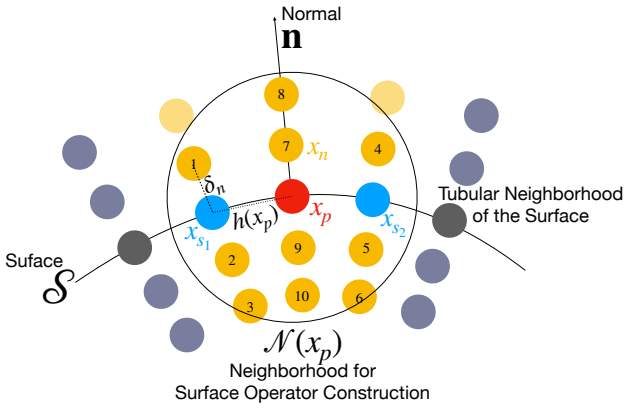



Figure 3.6: Illustration of the Surface DC-PSE method. The intrinsic differential operator at a point $\boldsymbol{x}_p$ (red) on the surface $\mathcal{S}$ is evaluated over neighboring surface points $\boldsymbol{x}_s$ (blue). The operator kernel is constructed in the tubular neighborhood with points $\boldsymbol{x}_n$ (yellow) replicated along the local surface normal $\boldsymbol{n}$ at distances $\delta n$. Point numbers and labels are used in the main text for the illustrative example.

### 3.6.2 Discretization of surface differential operators

We generalize DC-PSE to surface differential operators based on the following classic result [165, 169]: Let $\mathcal{S} \subset \mathbb{R}^d$ be a differentiable manifold that possesses a tubular neighborhood $T$ and is orientable[4] and $f : \mathcal{S} \to \mathbb{R}$. Define $F : T \to \mathbb{R}$, such that the restriction $F|_{\mathcal{S}} = f$, and $F$ is constant along the normal direction $\boldsymbol{n}$ of $\mathcal{S}$, i.e., $\nabla F \cdot \boldsymbol{n} = 0$. Then, on the surface $\mathcal{S}$,

$$\nabla_{\mathcal{S}} f = (\nabla F)|_{\mathcal{S}}, \tag{3.23}$$

[4]Every boundary-less smooth surface embedded in $\mathbb{R}^d$ has a tubular neighborhood, and the orientability condition is not restrictive when considered locally [169].

where $\nabla_{\mathcal{S}} f$ is the intrinsic surface gradient. A similar result is true for the intrinsic divergence operator $(\nabla_{\mathcal{S}} \cdot\ )$ and for tangential vector field that is extended by constant extension to all surfaces displaced along the normal of $\mathcal{S}$ [165, 169].

Given this result, it is straightforward to see the advantages of a mesh-free discretization: it allows for conforming discretization of the surface and for exact constant orthogonal extension by simply copying points along the normal. This creates an embedding narrow-band of exact closest-point function values within the tubular neighborhood $T$ without a need for interpolation. If $T$ has a radius of at least $r_c$ everywhere, the conditions of the result in Eq. (3.23) are satisfied in the constructed embedding. Analogous to the closest-point method [165], one can then discretize differential operators in the embedding space. Due to the additive kernel nature of DC-PSE, the discrete operators in the embedding space can be reduced to only the surface points.

This reduction becomes clear from the formulation of the DC-PSE method. Indeed, we realize that the constant normal extension can be made internal to the operator evaluation by accumulating the kernel *coefficients* along the normals. To see this, consider the DC-PSE operator in Eq. (3.10) in the embedding space. The neighborhood $\mathcal{N}$ for the summation contains both surface points $\boldsymbol{x}_s$ and normally extended points $\boldsymbol{x}_n$, as shown in Fig. 3.6b. Because the $f(\boldsymbol{x}_n)$ are identical copies of the values of the respective surface points, we note that the pre-factors $\big(f(\boldsymbol{x}_s) \pm f(\boldsymbol{x}_p)\big)$ in the summation of Eq. (3.10) are the same for all extended normal points and the corresponding surface point $\boldsymbol{x}_s$. Hence, for each given pair of a center point $\boldsymbol{x}_p$ and another surface point $\boldsymbol{x}_s$, the interactions with the corresponding normally extended points can be factored out from the kernel summation:

$$\frac{f(\boldsymbol{x}_s) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)^{|\alpha|}} \sum_{\boldsymbol{x}_q = \{\boldsymbol{x}_s, \boldsymbol{x}_n : (\boldsymbol{x}_n - \boldsymbol{x}_s) || \boldsymbol{n}(\boldsymbol{x}_s)\}} \eta_\epsilon^p \left( \frac{\boldsymbol{x}_p - \boldsymbol{x}_q}{\epsilon\left(\boldsymbol{x}_p\right)} \right) = \frac{f(\boldsymbol{x}_s) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)^{|\alpha|}} \eta_{\mathcal{S}}(\boldsymbol{x}_p, \boldsymbol{x}_s), \quad (3.24)$$

defining the surface kernels $\eta_{\mathcal{S}}(\boldsymbol{x}_p, \boldsymbol{x}_s)$. These can be evaluated over only the surface points $\boldsymbol{x}_s = \mathcal{N}_{\mathcal{S}}(\boldsymbol{x}_p)$ in the in-surface neighborhood $\mathcal{N}_{\mathcal{S}}(\boldsymbol{x}_p)$ around the surface point $\boldsymbol{x}_p$, see Fig. 3.6b, yielding the Surface DC-PSE operator:

$$\boldsymbol{Q}^{\boldsymbol{\alpha}}_{\mathcal{S}} f(\boldsymbol{x}_p) = \frac{1}{\epsilon(\boldsymbol{x}_p)^{|\alpha|}} \sum_{\boldsymbol{x}_s \in \mathcal{N}_{\mathcal{S}}(\boldsymbol{x}_p)} \big(f(\boldsymbol{x}_s) \pm f(\boldsymbol{x}_p)\big)\, \eta_{\mathcal{S}}(\boldsymbol{x}_p, \boldsymbol{x}_s). \quad (3.25)$$

Importantly, the surface kernels $\eta_{\mathcal{S}}(\boldsymbol{x}_p, \boldsymbol{x}_s)$, summed over all orthogonally extended points, can directly be computed when determining the kernel weights and without explicitly creating or storing the normally extended points $\boldsymbol{x}_n$.

Evaluating a Surface DC-PSE operator involves only the neighboring points on the surface and requires no narrow band or normally extended grid, even though the construction of the operators uses an embedding. This leads to a corresponding reduction in computational complexity for operator evaluation, as computations are only performed on a $(d-1)$-dimensional surface embedded in $d$-dimensional space. In comparison, the

cost of operator evaluation for embedding methods such as the closest-point method is $O(k(d-1))$, where $k > 1$ is the narrow-band width, which scales proportionally with the order of convergence.

### 3.6.3 Surface DC-PSE kernel construction

Surface DC-PSE requires two algorithms that are not part of the standard, flat-space DC-PSE method: an algorithm to create the intrinsic neighborhood of a surface point $p$, and an algorithm to determine the surface kernels $\eta_{\mathcal{S}}$ at a surface point $p$. We follow the example of Ref. [78] and use explicit component notation and a concrete example in order to directly relate to implementations in computer code.

Figure 3.6b illustrates a piece of the tubular neighborhood of a curved surface $\mathcal{S}$ embedded in $\mathbb{R}^2$. The red point $p$ at position $\boldsymbol{x}_p$ on the surface is the "center" collocation point at which we derive the discrete Surface DC-PSE operator $\boldsymbol{Q}^{\boldsymbol{\alpha}}_{\mathcal{S}} f(\boldsymbol{x}_p)$ for a scalar surface field $f$. Points in light blue are surface points within the embedding-space neighborhood (circle) of $\boldsymbol{x}_p$, and the yellow points are the orthogonal extensions $\boldsymbol{x}_n$. Only surface points are actually allocated and stored.

In order to determine the DC-PSE kernel $\eta^p_\epsilon$ in the embedding space, the distances between $\boldsymbol{x}_p$ and all collocation points in its embedding-space neighborhood $\mathcal{N}$ are required. In the example of Fig. 3.6b, the neighborhood (circle) includes the surface points $\mathcal{N}_{\mathcal{S}} = \{s_1, s_2\}$ and the normally extended points $\{n_i\}_{i=1}^{10}$. Algorithm 3.3 constructs this neighbor set along with the corresponding distances. Surface normals at a given point are indexed by the point index for better readability, i.e., $\boldsymbol{n}(\boldsymbol{x}_p) := \boldsymbol{n}_p$. In the example of the Fig. 3.6, this results in the output

$$\mathcal{N}_{\text{dist}}(\boldsymbol{x}_p) = [[\boldsymbol{d}_{s_1}, \boldsymbol{d}_{n_1}, \boldsymbol{d}_{n_2}, \boldsymbol{d}_{n_3}], [\boldsymbol{d}_{s_2}, \boldsymbol{d}_{n_4}, \boldsymbol{d}_{n_5}, \boldsymbol{d}_{n_6}], [\boldsymbol{d}_{n_7}, \boldsymbol{d}_{n_8}, \boldsymbol{d}_{n_9}, \boldsymbol{d}_{n_{10}}]],$$

where $\boldsymbol{d}_q$ is the distance between the collocation points $p$ and $q$ in the embedding space in units of $\epsilon_p := \epsilon(\boldsymbol{x}_p)$.

Using this neighborhood data structure, the embedding-space DC-PSE operator at point $p$ in the example of the figure reads:

$$\begin{aligned}
\boldsymbol{Q}^{\boldsymbol{\alpha}}_{\mathcal{S}} f(\boldsymbol{x}_p) =& \frac{f(\boldsymbol{x}_{s_1}) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)} \left(\eta^p_\epsilon(\boldsymbol{d}_{s_1}) + \eta^p_\epsilon(\boldsymbol{d}_{n_1}) + \eta^p_\epsilon(\boldsymbol{d}_{n_2}) + \eta^p_\epsilon(\boldsymbol{d}_{n_3})\right) \\
&+ \frac{f(\boldsymbol{x}_{s_2}) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)} \left(\eta^p_\epsilon(\boldsymbol{d}_{s_2}) + \eta^p_\epsilon(\boldsymbol{d}_{n_4}) + \eta^p_\epsilon(\boldsymbol{d}_{n_5}) + \eta^p_\epsilon(\boldsymbol{d}_{n_6})\right) \\
&+ \frac{f(\boldsymbol{x}_p) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)} \left(\eta^p_\epsilon(\boldsymbol{d}_{n_7}) + \eta^p_\epsilon(\boldsymbol{d}_{n_8}) + \eta^p_\epsilon(\boldsymbol{d}_{n_9}) + \eta^p_\epsilon(\boldsymbol{d}_{n_{10}})\right). \qquad (3.26)
\end{aligned}$$

Since the embedding-space kernels are evaluated at concrete distances, the $\eta^p_\epsilon(\boldsymbol{d}_q)$ are just scalar numbers. All kernel values that share the same pre-factor $\frac{f(\boldsymbol{x}_q) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)}$ are thus

**Algorithm 3.3** Surface DC-PSE: construction of neighborhood of a point $p$.
**Input:**

1. Point set $\mathsf{P}$ on the surface $\mathcal{S}$
2. Cutoff radius for the operator support $r_c$
3. Indices $\mathcal{N}_\mathcal{S}$ of surface points in the neighborhood of $p$
4. Number of normal copies of each surface point to be used during operator construction $N_n$ (symmetric to both sides of the surface)
5. Optional: spacing $\delta n$ between the normally extended points. Default: average embedding-space distance $h$ between surface points

**Output:** List of distances between point $p$ and all surface $s_i$ and normal $n_i$ points in its neighborhood $\mathcal{N}(\boldsymbol{x}_p)$: $\mathcal{N}_{\text{dist}}(\boldsymbol{x}_p)$

**Require:** $|\boldsymbol{n}_{s_i}| = |\boldsymbol{n}_p| = 1$
$\mathcal{N}_{\text{dist}} = [\,]$, $k = 0$
**for all** $s_i \in \mathcal{N}_\mathcal{S}$ **do**
    $\mathcal{N}_{\text{dist}}$.append([ ])
    **for** $i \in [-N_n, N_n]$ **do**
        $\boldsymbol{d}_{n_i} = \boldsymbol{x}_p - \boldsymbol{x}_{s_i} - i\delta n \cdot \boldsymbol{n}_{s_i}$
        **if** $|\boldsymbol{d}_{n_i}| \leq r_c$ **then**
            $\mathcal{N}_{\text{dist}}[k]$.append($\boldsymbol{d}_{n_i}/\epsilon_p$) ▷ Add to set of the corresponding $s_i$.
    $k$ += 1

$\mathcal{N}_{\text{dist}}$.append([ ])
**for** $i \in [-N_n, N_n]\backslash\{0\}$ **do** ▷ Normal points to $p$.
    $\boldsymbol{d}_{n_i} = -i\delta n \cdot \boldsymbol{n}_p$
    **if** $|\boldsymbol{d}_{n_i}| \leq r_c$ **then**
        $\mathcal{N}_{\text{dist}}[k]$.append($\boldsymbol{d}_{n_i}/\epsilon_p$) ▷ Create a new set containing all points along $\boldsymbol{n}_p$.

summed to the surface kernels $\eta_\mathcal{S}(\boldsymbol{x}_p, \boldsymbol{x}_q)$, $q \in \mathcal{N}_\mathcal{S}$, obtaining:

$$\begin{aligned} \boldsymbol{Q}^{\boldsymbol{\alpha}}_\mathcal{S} f(\boldsymbol{x}_p) =& \frac{f(\boldsymbol{x}_{s_1}) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)}\, \eta_\mathcal{S}(\boldsymbol{x}_p, \boldsymbol{x}_{s_1}) + \frac{f(\boldsymbol{x}_{s_2}) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)}\, \eta_\mathcal{S}(\boldsymbol{x}_p, \boldsymbol{x}_{s_2}) \\ &+ \frac{f(\boldsymbol{x}_p) \pm f(\boldsymbol{x}_p)}{\epsilon(\boldsymbol{x}_p)}\, \eta_\mathcal{S}(\boldsymbol{x}_p, \boldsymbol{x}_p). \end{aligned} \tag{3.27}$$

For even differential operators, i.e., derivatives with even $|\boldsymbol{\alpha}|$, the third term vanishes

identically and can be skipped in the calculations. But this is not the case for odd-order derivatives. With this rearrangement, each evaluation of the operator at a point $p$ only requires three kernel evaluations instead of the 12 that would be required in the embedding case. In addition, the normally extended points never need to be allocated and stored, as all kernel computations can happen on the fly. Algorithm 3.4 details the procedure for Surface DC-PSE operator construction. For the example from Fig. 3.6b, this results in the surface DC-PSE kernel values:

$$\mathcal{K}_{\mathcal{S}} = [\eta_{\mathcal{S}}(\boldsymbol{x}_p, \boldsymbol{x}_{s_1}),\ \eta_{\mathcal{S}}(\boldsymbol{x}_p, \boldsymbol{x}_{s_2}),\ \eta_{\mathcal{S}}(\boldsymbol{x}_p, \boldsymbol{x}_p)],$$

which can directly be used in Eq. (3.25) to evaluate surface differential operators.

**Algorithm 3.4** Surface DC-PSE: construction of surface kernel at a point $p$.

**Input:**

1. List of neighbor distances $\mathcal{N}_{\text{dist}}$ for each pair $\boldsymbol{x}_{p,q}$, with $q \in \mathcal{N}(\boldsymbol{x}_p)$, as constructed by Algorithm 3.3

**Output:** Surface kernel values for each pair $\boldsymbol{x}_{p,s}$, with $s \in \mathcal{N}_{\mathcal{S}}$: $\mathcal{K}_{\mathcal{S}}$

$\mathcal{K}_{\mathcal{S}} = zeros(size(\mathcal{N}_{\mathcal{S}}) + 1)$
$\eta = \text{DCPSE}(p)$ ▷ Determine embedding-space DC-PSE kernel at $p$
**for all** $(i, j) \in \mathcal{N}_{\text{dist}}$ **do**
    $\mathcal{K}_{\text{emb}}[i][j] = \eta(\mathcal{N}_{\text{dist}}[i][j])$
**for** $i \in [1, size(\mathcal{N}_{\mathcal{S}}) + 1]$ **do**
    **for all** $j \in \mathcal{K}_{\text{emb}}[i]$ **do**
        $\mathcal{K}_{\mathcal{S}}[i]$ += $\mathcal{K}_{\text{emb}}[i][j]$

### 3.6.4 Validation and applications of Surface DC-PSE

In this section, we validate and benchmark the Surface DC-PSE method. First, we verify its convergence in test cases with known analytical solution. Then, we show applications to cases with more general surfaces where no analytical solution is available.

#### Laplace-Beltrami operator on a circle and a sphere

We start by verifying convergence for the Laplace-Beltrami operator on the unit circle $S^1$. The collocation points are distributed regularly using equi-angular spacing. We use a normal spacing of $\delta n = 3/(N_p - 1)$ to compute the surface operators in Eq. (3.25) in a narrow band with $N_n = 4$ layers on each side of the surface. $N_p$ is the number of surface points $\boldsymbol{x}_s$. We

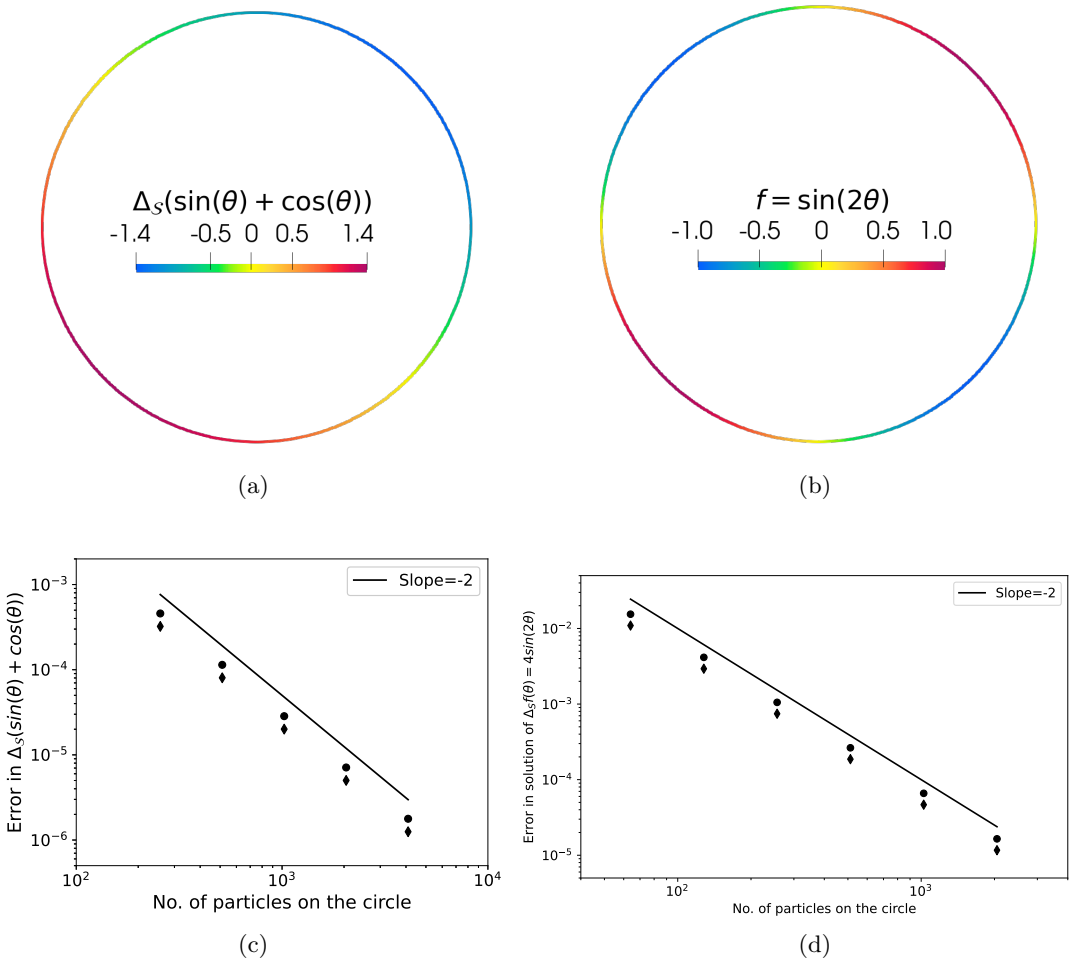



Figure 3.7: **Visualization and convergence of the Laplace-Beltrami operator on the unit circle. (a)** Visualization of $\Delta_{\mathcal{S}}(\sin(\theta)+\cos(\theta))$ computed using second-order accurate Surface DC-PSE operators. **(b)** Visualization of the solution of the Poisson equation $\Delta_{\mathcal{S}} f = 4\sin(2\theta)$ solved using second-order accurate Surface DC-PSE operators. **(c)** Convergence plot of the Laplace-Beltrami operator in (a). $L_\infty(\bullet)$ and $L_2(\blacklozenge)$ norms of the absolute errors are computed against the analytical solution in Eq. (3.29) for increasing numbers of points on the circle. **(d)** Convergence plot of the Poisson equation solution in (b). $L_\infty(\bullet)$ and $L_2(\blacklozenge)$ norms of the absolute errors are computed against the analytical solution in Eq. (3.33) for increasing numbers of points on the circle.

choose $r_c = 4.1\delta n$ as the operator support and $r = 2$ as the desired order of convergence. The Laplace-Beltrami operator is characterized by the multi-index $\boldsymbol{\alpha} = (2,0)+(0,2)$. Note

that this multi-index is 2-dimensional, despite the circle being one-dimensional, since the operators are constructed in the embedding space, but evaluated intrinsically.

We test the numerical approximation of the surface operator on the function

$$f(\theta) = \sin(\theta) + \cos(\theta) \tag{3.28}$$

in polar coordinates. The error is computed against the analytical solution

$$\Delta_{\mathcal{S}} f(\theta) = \nabla_{\mathcal{S}} \cdot (\nabla_{\mathcal{S}} f(\theta)) = \nabla_{\theta}^2 f(\theta) = -(\sin(\theta) + \cos(\theta)). \tag{3.29}$$

The visualization of the numerical solution and the convergence plot of the absolute errors are shown in Fig. 3.7a,c. We observe second-order convergence to the analytical solution, as expected for $r = 2$.

We further test the method on the unit sphere $S^2$. The collocation points are distributed using the Fibonacci sphere technique [170]. We use a normal spacing of $\delta n = 0.8/(\sqrt[3]{N_p} - 1)$ to determine the surface operators in Eq. (3.25) in a narrow band with $N_n = 2$ layers on each side of the surface. $N_p$ is the number of points on the sphere. We choose $r_c = 2.9\delta n$ as the operator support and $r = 2$ and $r = 4$ as the desired orders of convergence. The Laplace-Beltrami operator is characterized by the three-dimensional multi-index $\boldsymbol{\alpha} = (2, 0, 0) + (0, 2, 0) + (0, 0, 2)$. We test the numerical approximation of the surface operator on the scalar spherical harmonic function

$$f(\theta, \phi) = Y_{l,m} \tag{3.30}$$

in spherical coordinates for the mode $l = 4$, $m = 0$ (Fig. 3.8a). The error is computed against the analytical solution

$$\Delta_{\mathcal{S}} f = \nabla_{\mathcal{S}} \cdot (\nabla_{\mathcal{S}} f(\theta, \phi)) = -l(l+1) Y_{l,m} \tag{3.31}$$

and is plotted in Fig. 3.8c.

We also use this test case to benchmark against the Closest Point (CP) method [165] with $L_2$ and $L_\infty$ errors plotted in Fig. 3.8c. While Surface DC-PSE is less accurate than the CP method for second-order operators, it outperforms for fourth-order operators, which is likely due to the better condition numbers of the small linear systems to be solved for each point.

Finally, we perform a strong scaling benchmark of the computation time with increasing numbers of CPU cores with both codes implemented in the parallel computing library OpenFPM [64, 102] in C++ and run on the same hardware. We plot the parallel efficiency (i.e., the speed-up divided by the number of cores) in Fig. 3.8b for both the construction and the evaluation of the operators of both methods. We further report the absolute wall-clock times on one and 24 cores in the inset table. These times show that evaluating the Surface DC-PSE operators over all points in the domain is about one order of magnitude faster than evaluating the CP transform [165]. In addition, Surface DC-PSE scales better with

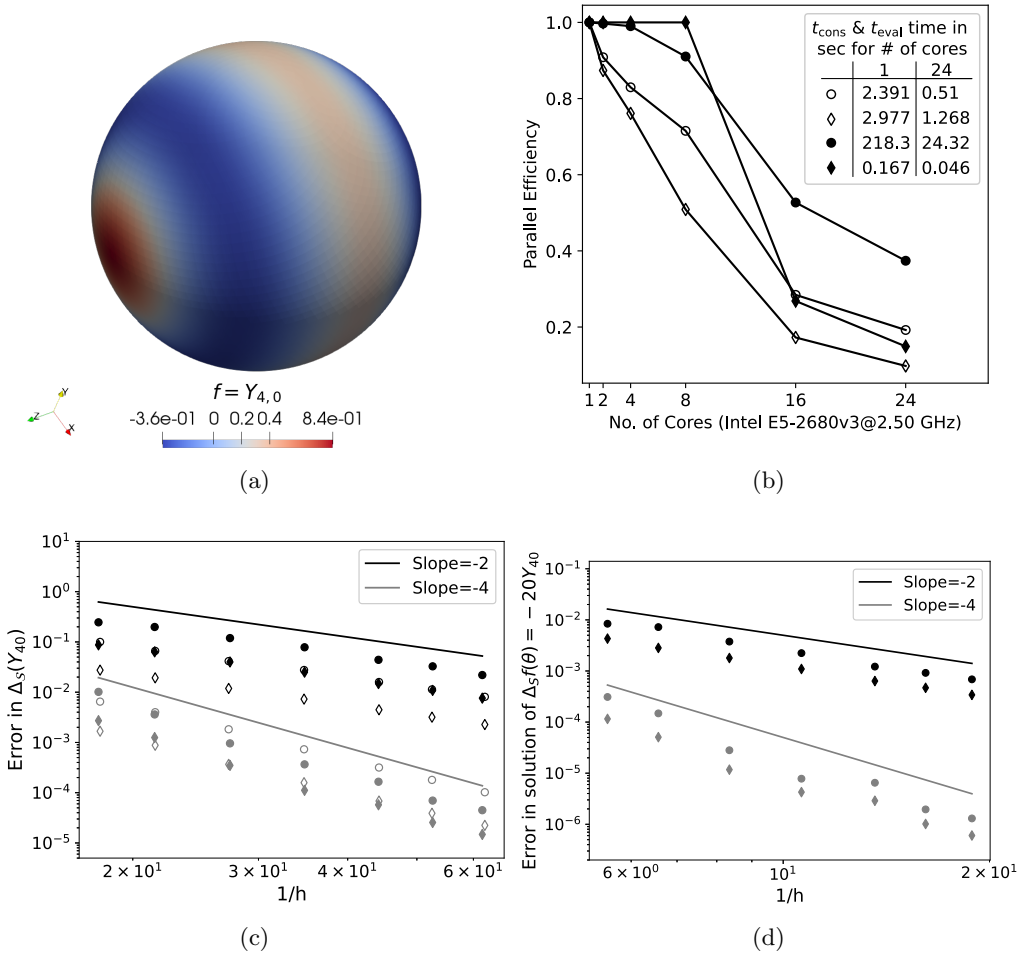



Figure 3.8: **Visualization and convergence of the Laplace-Beltrami operator on the unit sphere. (a)** Visualization of the spherical harmonic function $Y_{4,0}$. **(b)** Results of a strong scalability test for the computation of $\Delta_{\mathcal{S}} Y_{4,0}$ for average spacing $h = 0.05$ using Surface DC-PSE operators ($\bullet, \blacklozenge$) and the Closest Point (CP) method ($\circ, \lozenge$) [165] on increasing numbers of CPU cores. We separately show the parallel efficiency for the construction of Surface DC-PSE operators ($\bullet$) and of the CP representation ($\circ$), and for their evaluation to approximate $\Delta_{\mathcal{S}} Y_{4,0}$ across the entire domain using Surface DC-PSE ($\blacklozenge$) and the CP method ($\lozenge$). The absolute wall-clock times for one time step are shown in the inset table for 1 and 24 cores. **(c)** Convergence plot of $\Delta_{\mathcal{S}} Y_{4,0}$ for Surface DC-PSE ($L_\infty(\bullet)$, $L_2(\blacklozenge)$) and the CP method ($L_\infty(\circ)$, $L_2(\lozenge)$) using second-order (black) and fourth-order (gray) approximations. Norms of the absolute errors are computed against the analytical solution in Eq. (3.31) for increasing numbers of points (decreasing average spacing $h$). **(d)** Convergence plot for the solution of the Poisson equation $\Delta_{\mathcal{S}} f = -20 Y_{4,0}$ across the entire domain using second-order (black) and fourth-order (gray) operators. $L_\infty(\bullet)$ and $L_2(\blacklozenge)$ norms of the absolute errors are computed against the analytical solution in Eq. (3.35) for increasing numbers of surface points.

increasing numbers of parallel CPU cores. This is due to the simpler kernel evaluation of DC-PSE[5]. Eventually, the efficiency for both methods drops, as is expected for strong scaling with constant communication overhead. The time required to construct the Surface DC-PSE operator kernels, however, is about two orders of magnitude larger than that for constructing the CP representation in a narrow band. However, it still scales better with increasing CPU core count. For Eulerian simulations, where collocation points do not move, the kernels have to be determined only once at the beginning of a simulation, or they can be loaded from files for standard point distributions, potentially leading to an overall lower computational cost of Surface DC-PSE.

**Poisson equation on a circle and a sphere**

Surface DC-PSE operators can also be used for implicit equations by solving a linear system of equations with a system matrix constructed using the Surface DC-PSE operators. We test this by solving the Poisson equation on the unit circle $S^1$:

$$\Delta_{\mathcal{S}} f = 4\sin(2\theta) \qquad \theta \in \Omega = S^1 \backslash (1,0) \tag{3.32}$$

with Dirichlet boundary condition at one point $(1,0)$ conforming to the analytical solution

$$f(\theta) = \sin(2\theta) \qquad \theta \in \Omega \cup (1,0). \tag{3.33}$$

We use the same Surface DC-PSE operators as in the previous subsection to construct the system of equations, which is then solved using the KSPGMRES solver from PETSc [98]. Fig. 3.7b,d show the solution $f$ and the convergence plot of the absolute error with respect to the analytical solution.

Next, we test the method in three dimensions by solving the Poisson equation on the sphere $S^2$:

$$\Delta_{\mathcal{S}} f = -20 Y_{4,0} \tag{3.34}$$

with Dirichlet boundary condition along the great circle parallel to the $y-z$ plane conforming to the analytical solution

$$f = Y_{4,0}. \tag{3.35}$$

We solve the resulting linear system with KSPGMRES from PETSc [98] without preconditioning. The convergence plots for orders $r = 2$ and $r = 4$ are shown in Fig. 3.8d. The collocation points on the sphere were constructed using the Fibonacci sphere technique [170]. While this generates pseudo-regular point distributions on the sphere, their average spacing does fluctuate a bit, explaining the slight waviness of the curve especially for the fourth-order operators.

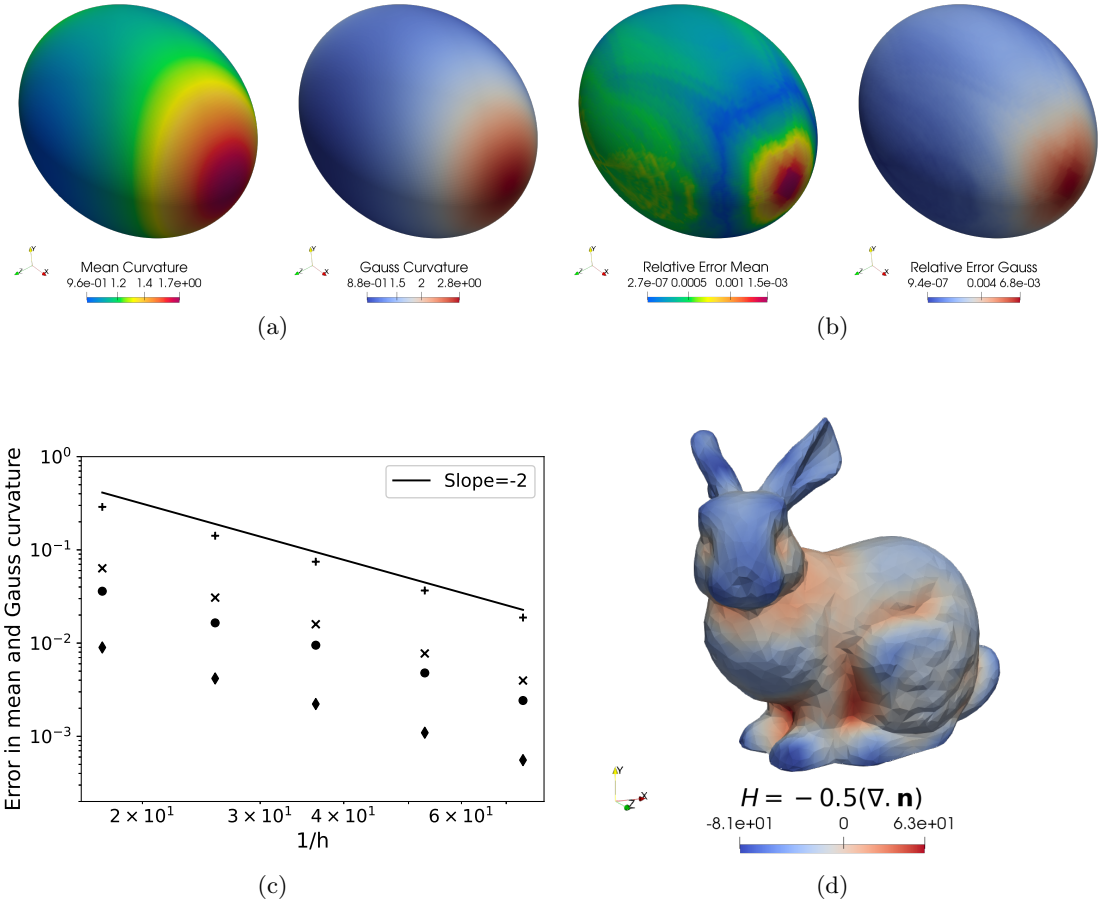



Figure 3.9: **Gauss and mean curvature computation using Surface DC-PSE. (a)** Visualization of the mean and Gauss curvatures of an ellipsoid, numerically computed as the trace of the embedded shape tensor $-0.5\nabla_{\mathcal{S}} \cdot \boldsymbol{n} = -0.5\mathrm{Tr}(\nabla_{\mathcal{S}}\boldsymbol{n})$ of the surface normal vector field $\boldsymbol{n}$ and the product of the non-zero eigenvalues of $\nabla_{\mathcal{S}}\boldsymbol{n}$, respectively, using second-order accurate Surface DC-PSE operators. **(b)** Visualization of the relative errors in the computed curvatures from (a) on 32258 surface points in comparison with the analytical solution in Eq. (3.38). They range from about $10^{-7}$ to $10^{-3}$. **(c)** Convergence plot of the mean and Gauss curvature computations. $L_\infty(\bullet, +)$ and $L_2(\blacklozenge, \times)$ absolute errors for mean and Gauss curvatures, respectively, are computed against the analytical solutions in Eq. (3.38) for decreasing average collocation point spacing $h$. **(d)** Visualization of the mean curvature computed on the Stanford bunny with 2960 points using second-order accurate Surface DC-PSE operators.

**Mean and Gauss curvature computation**

We further verify Surface DC-PSE by computing the mean curvature $H$ and Gauss curvature $K$ of an ellipsoid

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1 \tag{3.36}$$

with $a = 1$, $b = 0.8$, $c = 0.75$ and parameterization $(u, v)$

$$x = a\cos u \sin v, \qquad y = b \sin u \sin v, \qquad z = c \cos v. \tag{3.37}$$

We compute both curvatures from the embedded shape tensor $\nabla_{\mathcal{S}}\boldsymbol{n}$, i.e., the extension of the intrinsic shape operator to the embedding space. We numerically compute this tensor in Cartesian coordinates as the $3 \times 3$ matrix

$$\nabla_{\mathcal{S}}\boldsymbol{n} = \begin{pmatrix} \nabla_{\mathcal{S}} n_1 \\ \nabla_{\mathcal{S}} n_2 \\ \nabla_{\mathcal{S}} n_3 \end{pmatrix},$$

where $\boldsymbol{n} = (n_1, n_2, n_3)$ are the components of the analytically given normal vectors at the collocation point. All intrinsic derivatives over the scalar fields $n_1, n_2, n_3$ are approximated by Surface DC-PSE operators. Mean curvature $H$ is then computed as the trace of the embedded shape tensor and Gauss curvature $K$ as the product of its non-zero eigenvalues (principal curvatures). These numerical computations are verified against the analytical solutions

$$\begin{aligned} H &= \frac{abc[3(a^2+b^2) + 2c^2 + (a^2+b^2-2c^2)\cos(2v) - 2(a^2-b^2)\cos(2u)\sin^2 v]}{8[a^2b^2\cos^2 v + c^2(b^2\cos^2 u + a^2 \sin^2 u)\sin^2 v]^{3/2}} \\ K &= \frac{a^2b^2c^2}{\left[a^2b^2\cos^2 v + c^2\left(b^2\cos^2 u + a^2\sin^2 u\right)\sin^2 v\right]^2}. \end{aligned} \tag{3.38}$$

We approximate the embedded shape tensor $\nabla_{\mathcal{S}}\boldsymbol{n}$ using Surface DC-PSE with $\delta n = 3.0/(N_p - 1)$, $r_c = 2.9\delta n$, $N_n = 2$, and $r = 2$. The results and the convergence plot of the absolute errors are shown in Fig. 3.9a,c. As specified by $r$, we observe second-order convergence to the analytical solution when decreasing the average spacing $h$ between the points. The relative errors are visualized in Fig. 3.9b. They concentrate around extremal points of the curvature, as expected.

Finally, we apply the same mean-curvature computation to an arbitrary surface with no analytical solution, the Stanford bunny from the Stanford Computer Graphics Laboratory. We use the down-sampled version of the original data set with 2960 points on the surface, obtained from `https://www.stlfinder.com/model/stanford-bunny-S4kAUsKI/3091553`. The result is visualized in Fig. 3.9d, showing that the Surface DC-PSE qualitatively works also for non-algebraic surfaces.

[5] Up to 8 cores, the measured parallel efficiency of Surface DC-PSE was larger than 1.0, due to cache effects when storing the kernel coefficients.

## 3.7 Conclusion

In this chapter, we have presented an algorithm that directly tackles the 3D active Ericksen-Leslie equations that model active polar fluids. The active Ericksen-Leslie model considered here does not have specific polarity terms in the Frank free energy, such as terms proportional to $\mathbf{p}^2$ or $\mathbf{p}^4$. However, under confinement, Dirichlet (anchored) boundary conditions effectively impose a polarity to the system. Further, the equations considered here are equivalent to evolving a nematic system of infinite size in the regimes where there is no defect nucleation. When defects appear, the present formulation with a vector Laplacian for relaxation of polarity only allows integer charge defects, which are allowed by a polar vector symmetry, instead of half integer charges possible with nematic symmetry in two dimensions. In three dimensions, we expect similar behavior but for defect lines instead of point charges.

Our numerical approach uses a hybrid particle-mesh method and demonstrates a successful endeavor in solving hydrodynamics of three-dimensional active fluids. A crucial component of our algorithm is the Discretization Corrected Particle Strength Exchange (DC-PSE), which enables arbitrary higher-order discretization of differential operators over irregular particle distributions and simplifies the imposition of boundary conditions in complex domains. We have shown that our scalable iterative pressure-correction solvers converge on both regular and irregular particle distributions with curved boundaries.

The use of the OpenFPM open-source framework and the previously described template expression system for PDEs significantly enhance the computational feasibility of 3D active matter problems. We have offered a comprehensive derivation of the governing hydrodynamic equations, a detailed explanation of the algorithm, and a performance analysis. The efficient parallel scaling of our approach is proven in the simulations, which successfully converge to analytical solutions in thick active films with complex boundary conditions. This evidence underlines the correctness and potential of our methodology to address a myriad of active-matter problems in complex 3D geometries with arbitrary boundary conditions.

The scalability of the proposed algorithm is largely determined by the scalability of the pressure correction solver, with the linear system solver the primary limiting factor. The Portable Extensible Toolkit for Scientific Computation (PETSc) serves as our reference point for evaluating parallel scalability. While the scalability of the method is inherently bound by the scalability of PETSc, the modularity of OpenFPM allows the integration of alternative libraries after the matrix has been distributed. This implies that a future solver with enhanced scalability can directly improve the parallel efficiency of our current approach. The modular design also allows modifications to the model equations and geometry without requiring changes to the numerical method's source code, providing a highly adaptable and versatile platform for varying requirements.

Moreover, we have proposed a mesh-free collocation method for consistently approximating intrinsic differential operators on curved surfaces. This scheme is rooted in

the DC-PSE method for discretizing differential operators on irregular point clouds in flat spaces [77]. We've verified the method using different test cases with known analytical solutions and compared the scalability and accuracy of Surface DC-PSE to the classic closest-point method [165].

Surface DC-PSE has certain limitations: the requirement of the normal field as an input, the numerical error limitation by the curvature of the represented surface, and the inherent limit to surfaces possessing a non-intersecting tubular neighborhood with a radius at least as large as the kernel radius everywhere. Also, the computation of Surface DC-PSE kernels is computationally demanding.

In conclusion, our work in this chapter introduced a mesh-free algorithm for solving active hydrodynamics in arbitrary 3D geometries. Further, we proposed a new incremental method termed Surface DC-PSE, that allows for consistent and efficient discretization of surface differential operators. Future work can use Surface DC-PSE to solve problems in a Lagrangian frame of reference involving moving and deforming surfaces, and the coupling of Surface DC-PSE with regular DC-PSE in the surrounding space.

# Chapter 4

# Active hydrodynamics in 3D channels

In this chapter, we delve deeper into the realm of three-dimensional active hydrodynamics, applying the mesh-free algorithm we presented so far in Chapter 3. We harness the template expression system to devise a scalable simulation code within the OpenFPM environment to solve the complex hydrodynamics of active polar fluids in 3D [64]. The methodology encompasses an analysis of the three-dimensional equations, revealing spontaneous flow transitions in 3D active polar fluids [171]. The findings also indicate the existence of a topological phase transition that occurs subsequently to the spontaneous flow transition.

We propose an analytical expression describing a critical activity measure, or equivalently - a confining length, which acts as a threshold above which the system manifests spontaneous flow instability. By comprehensively characterizing this instability, we provide insights into the emergent behavior, contingent on the confining boundary conditions and the nature of the active stress, whether contractile or extensile. We juxtapose our analytical findings with results from numerous numerical simulations spanning a broad spectrum of parameters.

Further into this chapter, we extend the focus to numerical study of the equations for further increased activity. We observe the departure from linear regime, paving the way for a foray into non-linear dynamical behaviors. A surge in activity reveals the existence of traveling wave regimes, pointing towards a Berezinskii-Kosterlitz-Thouless (BKT) type topological phase transition [65, 172, 173], characterized by the manifestation of closely linked vortex pairs.

Historically, the acclaimed two-dimensional XY model, portraying a bi-dimensional vector on a grid, demonstrated a unique topological phase transition, the appearance of tightly coupled vortices at low temperatures [174]. A rise in temperature, however, destabilizes this coupling, enabling the vortices to move independently with correlations that decay exponentially [174]. Our study, strikingly, uncovers a similar phenomenon in active matter. As we amplify the activity, we traverse through intermittent oscillatory

regimes and transition into a chaotic dynamical phase.

It is pertinent to clarify, however, that the comparisons drawn with the XY model are only conceptual. Our system bears a closer resemblance to the Heisenberg model, a three-dimensional lattice of 3D vectors [175]. This analogy, while providing a rich context for our findings, also introduces a level of complexity that underlines the importance of the next section: a comprehensive examination of the 3D Erickson-Leslie model with Lagrange multipliers that links our model to the Heisenberg model. But first, we describe previous works on the spontaneous flow transition.

## 4.1 Inception and previous works

Active fluids are out-of-equilibrium materials driven by energy injection at the microscopic scale [22, 65]. Active materials can have a polar or nematic alignment symmetry of the orientation vector field, and the constituents of the material allow it to generate contractile or extensile active stress. Prominent examples of active fluids are found in living matter across scales, from the cytoskeleton [59, 176] and tissues [177–179] to collective behavior in flocks [21]. The hydrodynamic theory of incompressible active polar fluids describes the dynamics of such active liquid crystals at long wavelengths. A key behavior of active polar fluids is their ability to generate spontaneous flow under confinement and sufficient active stress.

Numerous advancements have been made in understanding spontaneous active flows. A seminal work used kinetic theory and nonlinear continuum simulations to explore the collective dynamics in suspensions of self-propelled particles [180, 181]. This showed that aligned suspensions are inherently unstable to fluctuations, generalizing previous predictions [182]. It further revealed that isotropic suspensions are prone to instability due to active stress, leading to significant density fluctuations and efficient fluid mixing [180]. Similarly, another study investigated the stability of swimming bacteria in Newtonian mediums [183] . They focused on bacteria performing run-and-tumble motion, revealing that an instability occurs in suspensions of 'pushers' like *E. coli*. This instability originates from the intrinsic force dipoles of such bacteria, leading to a negative viscosity effect in certain conditions, which destabilizes the suspension [183]. Furthermore, the dynamics of cytoplasmic streaming in plant cells and other biological active fluid systems within total geometric confinement has been explored [184]. Such systems could exhibit a stable circulating state under confinement. This indicated that an activity threshold exists for spontaneous autocirculation with observed behaviors akin to defect separation in nematic liquid crystals [184]. These studies collectively contribute to a deeper understanding of active matter systems, demonstrating intricate dynamics and instabilities that arise due to the unique properties of self-propelled particles in biologically relevant active fluids.

It has been shown in two dimensions (2D) that spontaneous flow can emerge due to a Fréedericksz-type transition first observed in passive liquid crystals [185]. The

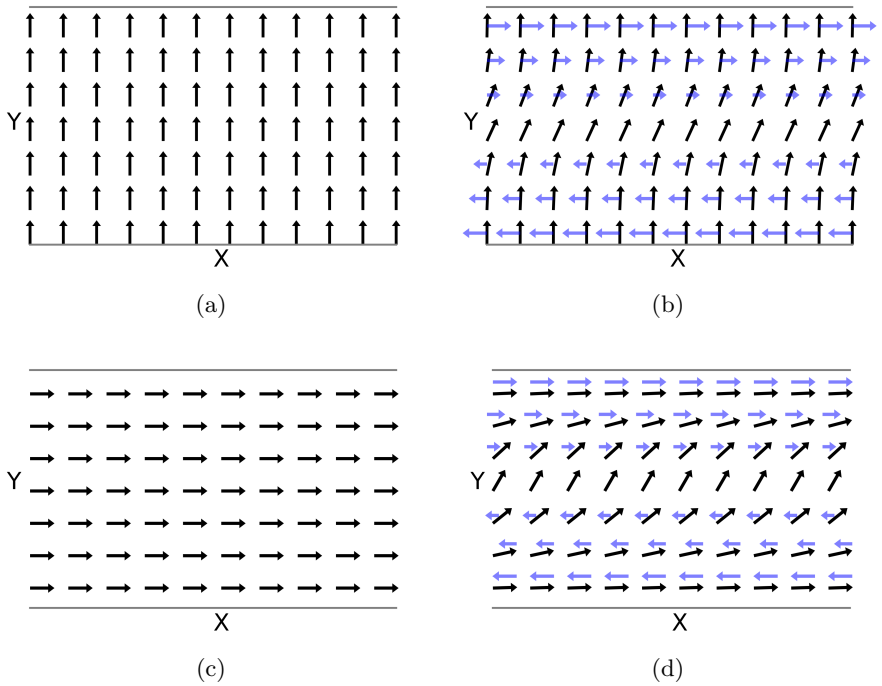
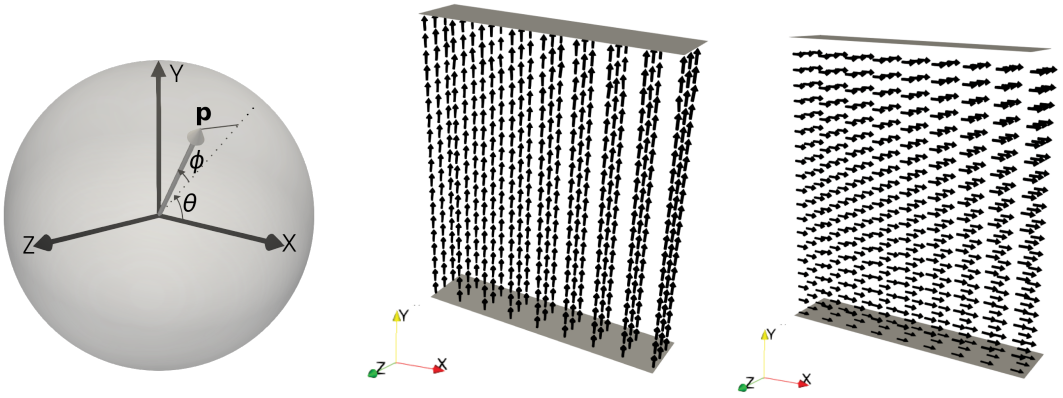



Figure 4.1: Two-dimensional active spontaneous flow transition in a plane that is infinite along X and of finite width $L$ along Y. **(a)** An initially homogeneous polarity field with perpendicular anchoring to the walls and no flow. **(b)** Spontaneous flow transition under *extensile* active stress with velocity indicated by blue arrows. **(c)** An initially homogeneous polarity field with parallel anchoring at the walls and no flow. **(d)** Transition to spontaneous flow under *contractile* active stress.

passive Fréedericksz transition describes the change of a homogeneous nematic state to an inhomogeneous state under the influence of external electric or magnetic fields [186]. In active liquid crystals, the transition is driven by active molecular processes causing spontaneous material flow [17, 61, 187–190]. In two dimensions, the type of instability depends on the confining boundary condition for the polarity field [176], as illustrated in Fig. 4.1. For polarity anchored perpendicular to boundaries, one obtains a spontaneous flow transition with extensile active stress (Fig. 4.1a,b) [17]. For polarity parallel to the boundary, the transition occurs for contractile active stress (Fig. 4.1c,d) [61, 188].

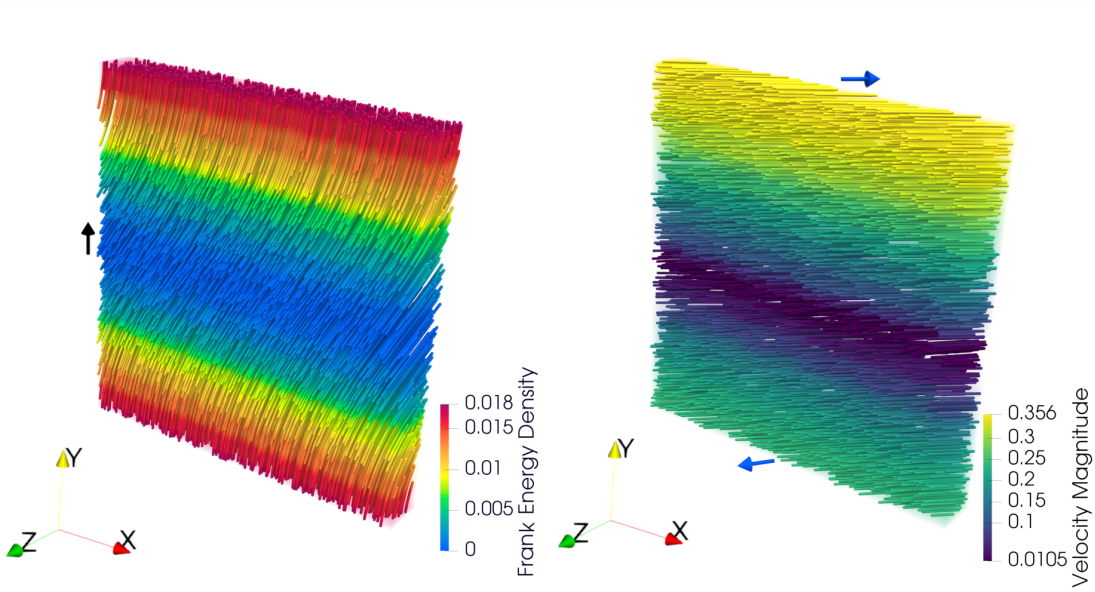
Recent works suggest such instabilities to also exist in three-dimensional (3D) active polar fluids [9, 43, 55, 191]. For example, an extensile active fluid was found to exhibit a bending instability in a minimal model [10], and flow-aligning active fluids were found to display coherent motion in 3D channels upon increased activity [192]. Furthermore, It has been shown that 3D contractile active fluids dampen out-of-plane perturbations, whereas




(a) (b) (c)

Figure 4.2: **(a)** Orientation of the unit polarity vector in three dimensions. $\theta$ is the angle between the polarity vector and the positive X axis, measured counterclockwise along the Y axis. $\phi$ is the angle between the polarity vector and the positive X axis measured clockwise along the Z axis. **(b)** No flow homogeneous steady state with perpendicular anchoring: Polarity vectors below the critical active potential ($\Delta\tilde{\mu} < \Delta\tilde{\mu}_c$). **(c)** No flow homogeneous steady state with parallel anchoring: Polarity vectors below the critical active potential ($\Delta\tilde{\mu} < \Delta\tilde{\mu}_c$).

extensile fluids amplify them in the absence of boundary effects [193]. 3D active fluids under confinement behave fundamentally different from their 2D counterparts. Notably, they exhibit flow due to buckling under extensile active stress, which is not possible in 2D for rigid boundaries [9]. Experimentally, such instabilities have been observed in microtubule assays capable of generating extensile active stresses [9, 55]. Some experiments suggested that in-plane and out-of-plane instabilities compete, depending on the material properties and the magnitude of the active stress [43, 191]. Other experiments [55] found only an out-of-plane instability. Currently, a theoretical model explaining and reconciling all observations is missing, and the dependence of the instability on the relevant system parameters and boundary conditions remains unexplained. Further, it is not clear from simplified models how the flow instability depends on system parameters like $\lambda$, $\zeta$, or $\nu$ [9, 10]. The microscopic origin of $\lambda$ and $\zeta$, however, can be elucidated in the full nonlinear model, providing experimentally testable predictions.

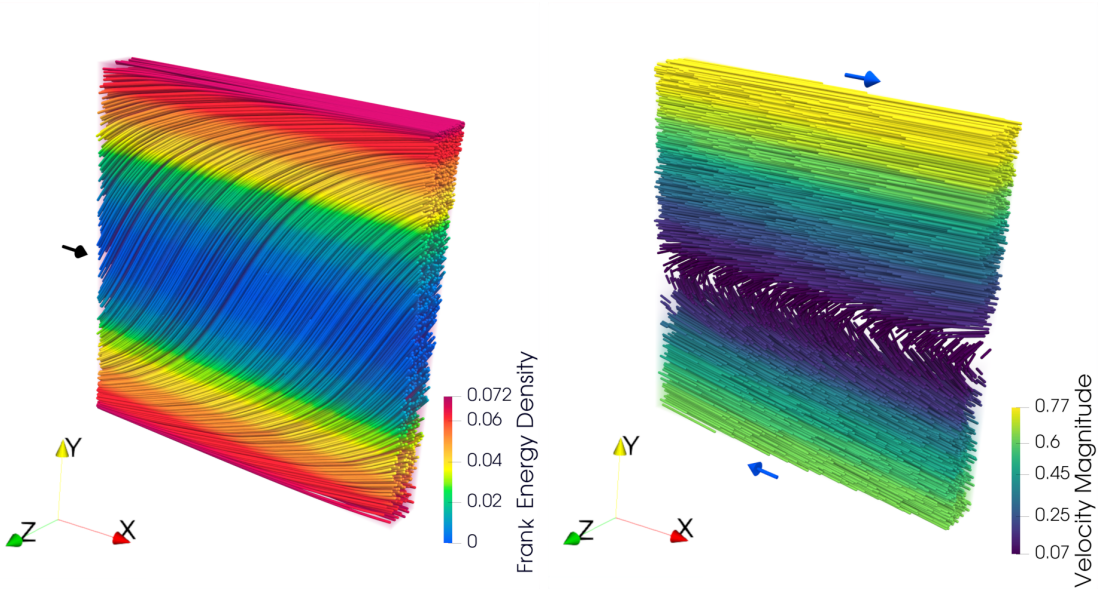



(a) (b)

Figure 4.3: Visualization of the 3D spontaneous flow transition for perpendicular polarity anchoring at the wall under extensile active stress. **(a) Polarity field in the spontaneous-flow steady state:** Polarity streamlines with non-zero polarity in both X and Z directions, and Frank free energy density as color. The black arrow indicates the initial polarity vector direction. **(b) Velocity field in the spontaneous-flow steady state:** Velocity streamlines for the spontaneous-flow steady state in (a) with non-zero flow in both X and Z directions, with velocity magnitude as color. The blue arrows indicate the directions of the flow at the stress-free walls.

## 4.2 Spontaneous flow in a 3D *Fréedericksz cell*

We study the governing equations (3.1) for a thick 3D active film that is infinitely long along the X and the Z directions and has a thickness of $L$ in the Y direction. We consider a finite domain with periodic boundary conditions in X and Z directions and wall boundaries in the Y direction. The surface of the fluid at $y = 0$ and $y = L$ is stress-free and impenetrable ($v_y = 0$). The polarity is fixed on the top and bottom by anchoring boundary conditions. At time zero, polarity is initialized everywhere homogeneous to the boundary condition, except for a perturbation of 0.001 radians in both $\theta$ and $\phi$ at $x = L/2, y = L/2$. For the hydrodynamic boundary conditions, we enforce stress-freeness ($\sigma^{\text{tot}}_{xy} = \sigma^{\text{tot}}_{yz} = 0$) at the $x = y = 0$ and $x = y = L$ planes and fix the integration constant by imposing zero flow velocity at $x = L/2$, $y = L/2$.

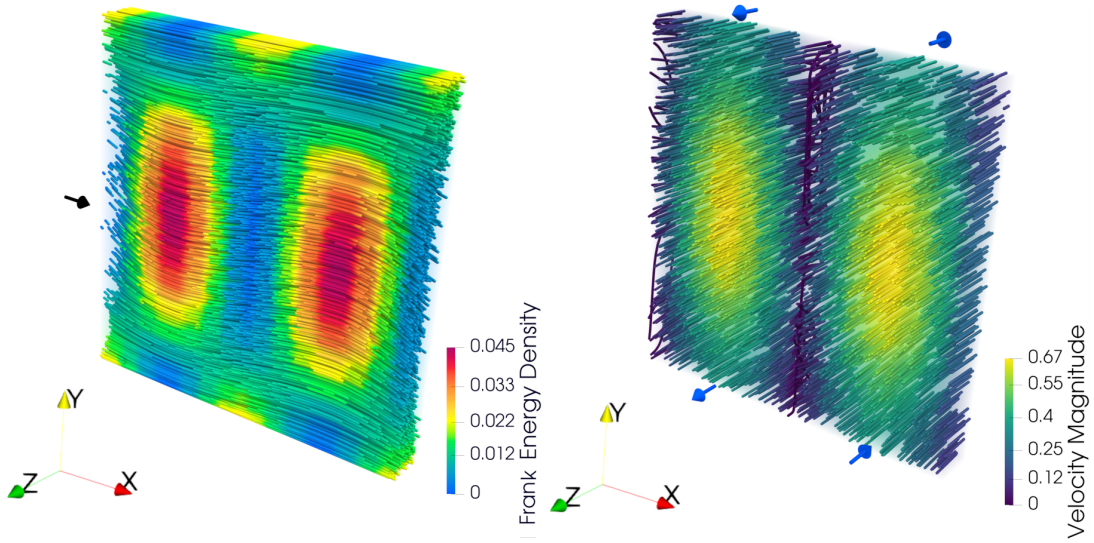



(a) (b)

Figure 4.4: Visualization of the 3D spontaneous flow transitions with parallel polarity anchoring at the wall and contractile active stress. **(a) In-plane spontaneous-flow steady state under contractile active stress:** Polarity streamlines showing non-zero polarity in Y direction and zero in Z direction, with Frank free energy density as color. The black arrow indicates the initial polarity vector direction. **(b) Velocity field in the spontaneous-flow steady state under contractile active stress:** Velocity streamlines for the in-plane spontaneous-flow steady state in (a) with non-zero flow in X direction and velocity magnitude as color. The blue arrows indicate the directions of the flow at the stress-free walls. Parameter values for (a-b) are: $\tilde{\gamma} = 1$, $\tilde{L} = 10$, $\tilde{\nu}$=-0.4 , $\tilde{\zeta} = -1$, $\Delta\tilde{\mu} = 0.35$.

### 4.2.1 Linear perturbation analysis

We analyze the hydrodynamic equations (3.1) at steady state and provide a mechanism for the emergence of symmetry-breaking spontaneous flow. For this, we express the unit polarity vector as $\mathbf{p} = [\cos(\theta)\cos(\phi), \sin(\theta)\cos(\phi), \sin(\phi)]$ using the coordinates illustrated in Fig. 4.2a. We eliminate the velocity from Eq. (3.1) and obtain the nonlinear force balance equation as a function of $(\theta, \phi)$. For anchoring boundary conditions $(\theta_0, \phi_0)$ at the top and bottom walls, the system is steady with trivial solution $(\theta(y), \phi(y)) = (\theta_0, \phi_0)$.




(a) (b)

Figure 4.5: Visualization of the 3D spontaneous flow transitions with parallel polarity anchoring at the wall and extensile active stress. **(a) Out-of-plane spontaneous-flow steady state under extensile active stress:** Polarity streamlines showing non-zero polarity field in Z direction and zero in the Y direction, with Frank free energy density as color. The black arrow indicates the initial polarity vector direction. **(b) Velocity in the out-of-plane spontaneous-flow steady state under extensile active stress:** Velocity streamlines for the spontaneous-flow steady state under extensile active stress in (b) with non-zero flow in the Z direction and velocity magnitude as color. The blue arrows indicate the directions of the flow at the stress-free walls. Parameter values for (a-b) are: $\tilde{\gamma} = 1$, $\tilde{L} = 10$, $\tilde{\nu}$=-0.4 , $\tilde{\zeta} = 1$ $\Delta\tilde{\mu} = 2.4$.

### Perpendicular anchoring

Fixing polarity to be perpendicular to the boundary wall, i.e., $(\theta_0, \phi_0) = (\pi/2, 0)$, and assuming small perturbations $\epsilon(y), \kappa(y)$, the restoring force up to linear order of tilt is $\mathbf{h}_\perp = (K\frac{\partial\kappa(y)}{\partial y^2}, 0, K\frac{\partial\epsilon(y)}{\partial y^2})$, where $K = K_s = K_t = K_b$ is the elastic constant in the single-constant approximation of the Frank free energy. Using Eq. (3.1c) and imposing $\sigma_{xy}^{(tot)} = \sigma_{zy}^{(tot)} = 0$, we obtain the strain rates $u_{xy}, u_{yz}$ and substitute them in Eq. (3.1a) to obtain the force associated with $\mathbf{h}_\perp$. The so-obtained non-linear equation is decoupled from the flow and only depends on $(\theta, \phi)$. We do not reproduce this equation here due to its excessive length. Substituting into this equation a small perturbation $(\epsilon(y), \kappa(y))$ and linearizing around $(\theta_0, \phi_0) = (\frac{\pi}{2}, 0)$, we obtain the dynamical equation of the perturbation

that leads to a spontaneous flow transition under extensile active stress,

$$K\frac{\partial^2}{\partial y^2}\begin{bmatrix}\epsilon(y)\\ \kappa(y)\end{bmatrix} = \frac{2\gamma\Delta\mu(\nu-1)(\zeta+\gamma\lambda\nu)}{\gamma(\nu-1)^2+4\eta}\begin{bmatrix}\epsilon(y)\\ \kappa(y)\end{bmatrix}. \tag{4.1}$$

Hence, there are perturbation modes of the form

$$\begin{bmatrix}\epsilon(y)\\ \kappa(y)\end{bmatrix} = \sin\left(\frac{\pi}{L}y\right)\begin{bmatrix}\epsilon_{\mathrm{m}}\\ \kappa_{\mathrm{m}}\end{bmatrix}, \tag{4.2}$$

where $\epsilon_{\mathrm{m}}, \kappa_{\mathrm{m}}$ are the maximum tilt magnitudes in $\theta$ and $\phi$, respectively. These modes effectively define a critical activity

$$\Delta\mu_c = \frac{K\pi^2\left(4\eta+\gamma(\nu-1)^2\right)}{2L^2\gamma(\nu-1)(-\zeta-\gamma\lambda\nu)}. \tag{4.3}$$

Assuming $-1 < \nu < 1$, i.e., a flow-tumbling active fluid, we note that for the critical $\Delta\mu_c > 0$ the effective active stress is extensile, $-\zeta - \gamma\lambda\nu < 0$. Hence for a flow-tumbling polarity, we expect the spontaneous flow transition when $\Delta\mu > \Delta\mu_c$. Up to linear orders of tilt, the strain rates in this case are:

$$\frac{\partial}{\partial y}\begin{bmatrix}v_x(y)\\ v_z(y)\end{bmatrix} = \frac{2\Delta\mu(\gamma\lambda\nu+\zeta)}{\gamma(\nu-1)^2+4\eta}\begin{bmatrix}-\epsilon(y)\\ \kappa(y)\end{bmatrix}. \tag{4.4}$$

However, the instability depends on a nonlinear combination of parameters. For example, the critical activity depends nonlinearly on $\nu$. Further, the effects of $\zeta$ and $\lambda$ are coupled to both the rotational viscosity $\gamma$ and to $\nu$. For $|\nu| > 1$, the behavior depends on $|\zeta|$. Three qualitatively different critical behaviors in Fig. 4.6.

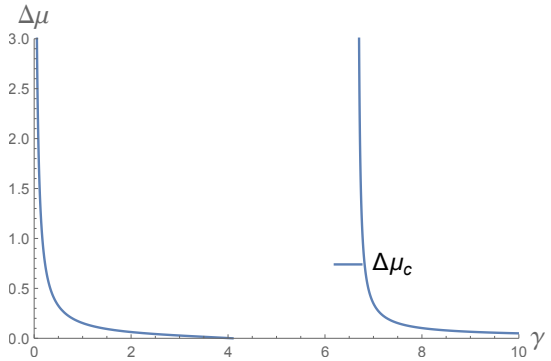
**Parallel anchoring**

We study the stability of parallel anchoring of the polarity at the boundary, i.e., $(\theta_0, \phi_0) = (0, 0)$. We repeat the analysis for $(\theta_0, \phi_0) = (0, 0)$ and obtain the critical activity of a 3D spontaneous flow transition under contractile active stress. In this case, the restoring force up to linear order of tilt is $\mathbf{h}_\perp = (0, -K\frac{\partial\kappa(y)}{\partial y^2}, K\frac{\partial\epsilon(y)}{\partial y^2})$. Up to linear order of tilt, the strain rate in this case is:

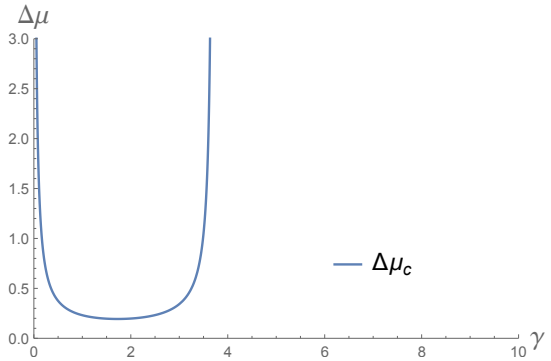
$$K\frac{\partial^2}{\partial y^2}\begin{bmatrix}\epsilon(y)\\ \kappa(y)\end{bmatrix} = \frac{2\gamma\Delta\mu(\nu+1)(\gamma\lambda\nu+\zeta)}{\gamma(\nu+1)^2+4\eta}\begin{bmatrix}\epsilon(y)\\ 0\end{bmatrix}. \tag{4.5}$$

This leads to a critical activity

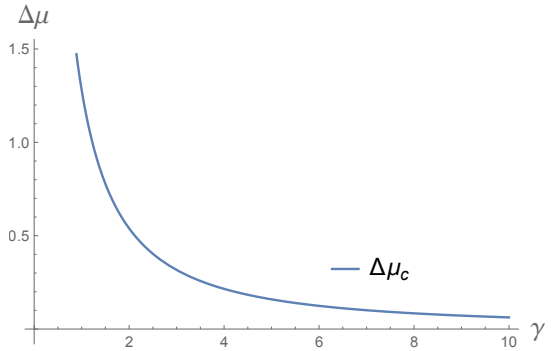
$$\Delta\mu_c = \frac{\pi^2 K\left(4\eta+\gamma(\nu+1)^2\right)}{2\gamma L^2(\nu+1)(-\zeta-\gamma\lambda\nu)}. \tag{4.6}$$



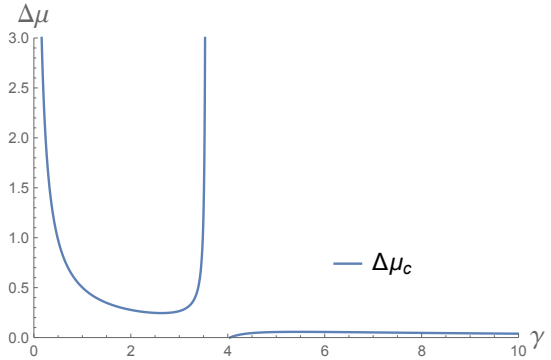

(a) Perpendicular anchoring with extensile stress. $\zeta = 1, \nu = -1.5$




(b) Parallel anchoring with contractile stress. $\zeta = -1, \nu = -0.27$




(c) Perpendicular anchoring with extensile stress. $\zeta = 1, \nu = 0.4$




(d) Perpendicular anchoring with extensile stress. $\zeta = 0.37, \nu = -0.1$

Figure 4.6: Dependence of the critical activity $\Delta\mu$ on the rotational viscosity $\gamma$.

For the critical $\Delta\mu_c > 0$, the active stress for the transition is contractile, $-\zeta - \gamma\lambda\nu > 0$ giving rise to a spontaneous flow transition with an S-like shape of the polarity field (Fig. 4.4a), as observed in 2D, but invariantly extended in the third dimension. This confirms that contractile active polar fluids impede out-of-plane perturbations when polarity is anchored on the boundary. The spontaneous flow transition occurs when $\Delta\mu > \Delta\mu_c$. Up to linear order of tilt, the strain rate in this case is:

$$\frac{\partial v_x(y)}{\partial y} = \frac{2\Delta\mu(\gamma\lambda\nu + \zeta)}{\gamma(\nu+1)^2 + 4\eta}\epsilon(y). \tag{4.7}$$

We note that the symmetry can also be broken in the X direction instead of Y, such that $u_{xz} \neq 0$ and $u_{xy} = 0$. Assuming $u_{xz} \neq 0$, $u_{yz} \neq 0$, but $u_{xy} = 0$, thus allowing polarity to vary in both the X and Y directions, $(\theta(x,y), \phi(x,y))$, and using $\mathbf{h}_\perp = (0, -K\nabla^2_{\{x,y\}}\kappa(x,y), 0)$, we find the two-dimensional perturbation mode that corresponds to the out-of-plane wrinkling transition under extensile active stress.

$$K\nabla^2_{\{x,y\}}\begin{bmatrix}\epsilon(x,y)\\ \kappa(x,y)\end{bmatrix} = \frac{-2\gamma\Delta\mu(\nu-1)(\gamma\lambda\nu+\zeta)}{\gamma\,(\nu-1)^2 + 4\eta}\begin{bmatrix}0\\ \kappa(x,y)\end{bmatrix}. \tag{4.8}$$

This leads to perturbations of the form

$$\kappa(x,y) = \kappa_m \cos\left(\frac{2\pi}{L_x}x + \alpha\right)\sin\left(\frac{\pi}{L_y}y\right), \tag{4.9}$$

where $\alpha$ is the phase shift, which is fixed by the flow boundary condition and the integration constant. This mode corresponds to a critical activity

$$\Delta\mu_c = \frac{(4L_x^2 + L_y^2)\pi^2 K\left(\gamma\,(\nu-1)^2 + 4\eta\right)}{2\gamma L_x^2 L_y^2(\nu-1)(-\zeta-\gamma\lambda\nu)}. \tag{4.10}$$

For the critical $\Delta\mu_c > 0$, the active stress for the transition is extensile, $-\zeta - \gamma\lambda\nu < 0$. In this case, the strain rate up to linear order of tilts is

$$\frac{\partial v_z(x,y)}{\partial x} = \frac{2\Delta\mu(\gamma\lambda\nu+\zeta)}{4\eta + \gamma\,(\nu-1)^2}\kappa(x,y). \tag{4.11}$$

The mode in Eq. (4.9) describes an out-of-plane transition maintaining $\theta(x,y) = 0$. For the previously chosen parameters and $\tilde{\zeta} = 1$, the dependence of $\Delta\tilde{\mu}_c$ on $\tilde{\nu}$ is shown in Fig. 4.7d.

The Eq. (4.10) further clarify the effect of the finite length $L_x$ of the domain in the X direction. In the ideal physical system, $L_x$ is infinite, and $L_y$ is finite. In Eq. (4.10), we see that there is a non-zero limit for the critical activity as $L_x$ approaches infinity, and the mode of deformation in Eq. (4.9) has no modulation in the X direction. This predicts the wrinkling wavelength close to the transition. For further increasing activity, however, the

perturbations are no longer small, rendering the linearized equations invalid. Then, the system transitions to spatiotemporal chaos.

The amplitudes $\epsilon_{\mathrm{m}}, \kappa_{\mathrm{m}}$ depend on $\Delta\tilde{\mu}$ and are analytically intractable. The critical active potential for out-of-plane wrinkling is significantly larger than for the other cases. This causes the instability to occur earlier or faster in time in 3D. The unstable mode shows oscillatory flows in opposite directions. With $\Delta\tilde{\mu} > \Delta\tilde{\mu}_c$, we find a spontaneous flow transition of small amplitude near the critical value, as shown by the maximum norm of the velocity (color of symbols). The associated wrinkling in the transition is shown in Fig. 4.5a-b. This transition has also been observed experimentally in extensile polar fluids and referred to as *bending* or *wrinkling instability* [9, 43, 55, 191]. Here, we qualitatively characterized the effect of finite channel length $L_x$ on the wrinkling wavelength. Further, our numerical simulations with changing the length scale qualitatively produce the predicted behavior.

### 4.2.2 Numerical simulations of spontaneous flow transition

The governing equations can be nondimensionalized with respect to $\lambda$, $\eta$, and $K$ by rescaling $\tilde{L} = (\eta\lambda)^{\frac{1}{3}}L$, $\tilde{\zeta} = \zeta(\eta\lambda)^{-1}$, $\Delta\tilde{\mu} = \Delta\mu(\eta\lambda)^{\frac{1}{3}}K^{-1}$, $\tilde{\gamma} = \gamma\eta^{-1}$, and $\tilde{\nu} = \nu$. We numerically solve the dimensionless equations for a thick 3D active film that is periodic along the X and the Z directions and has thickness $L$ in the Y direction. Details of the simulation method can be found in SI-2. The simulation computer code scales to parallel computer architectures, as it is based on the open-source scientific computing library OpenFPM [102] and a template expression language for partial differential equations [64].

We verify the expressions in Eq. (4.3) by numerically solving the nonlinear equations for $\tilde{\gamma} = 1$, $\tilde{\zeta} = 1$, $\tilde{L} = 10$. For these parameters, the dependence of the critical activity $(\Delta\tilde{\mu}_c)$ on the flow-tumbling parameter $\tilde{\nu}$ is shown in Fig. 4.7a as a solid line. Hence, when $\Delta\tilde{\mu} > \Delta\tilde{\mu}_c$, the mode in Eq. (4.2) appears with spontaneous flow governed by Eq. (4.4)

We find that the mode may be stabilized by the nonlinearities above the critical potentials, resulting in steady-state flows. In the steady flow state, active stresses are balanced by elastic nematic stresses, leading to stationary perturbation modes. The corresponding spontaneous flow transition is observed above the critical activity and is visualized in Fig. 4.2.

We confirm this in nonlinear simulations for the case of parallel anchoring same parameters as in the previous case and $\tilde{\zeta} = -1$ for contractile active stress. The dependence of the critical activity for contractile active stress on the flow-tumbling parameter $\tilde{\nu}$ is plotted in Fig. 4.7c as a solid line. The numerical solutions for $\Delta\tilde{\mu} > \Delta\tilde{\mu}_c$ confirm the transition for different values of $\nu$, the maximum norm of the flow velocity is shown as a color code in Fig. 4.7c.

We further perform simulations for the extensile active stress case $\tilde{\zeta} = -1$, with parallel anchoring. This confirms the 3D wrinkling instability of active fluids as observed in various experiments. The dependence of the critical activity for this case on the flow-tumbling

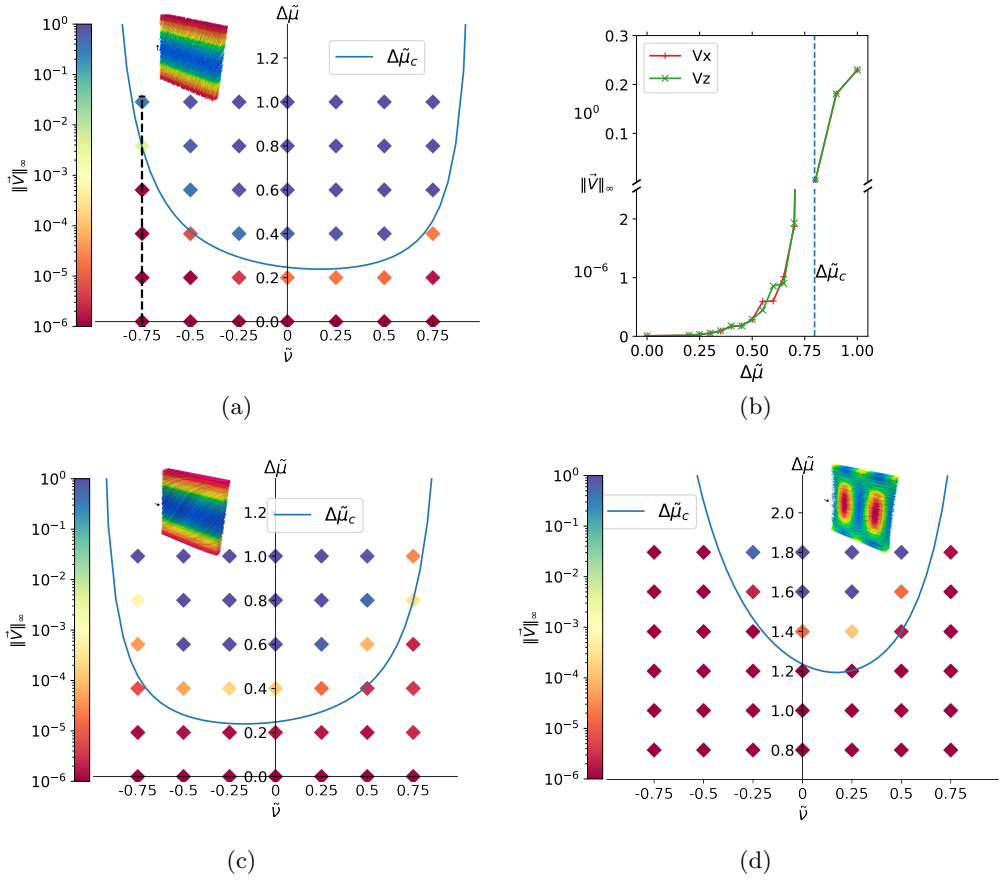
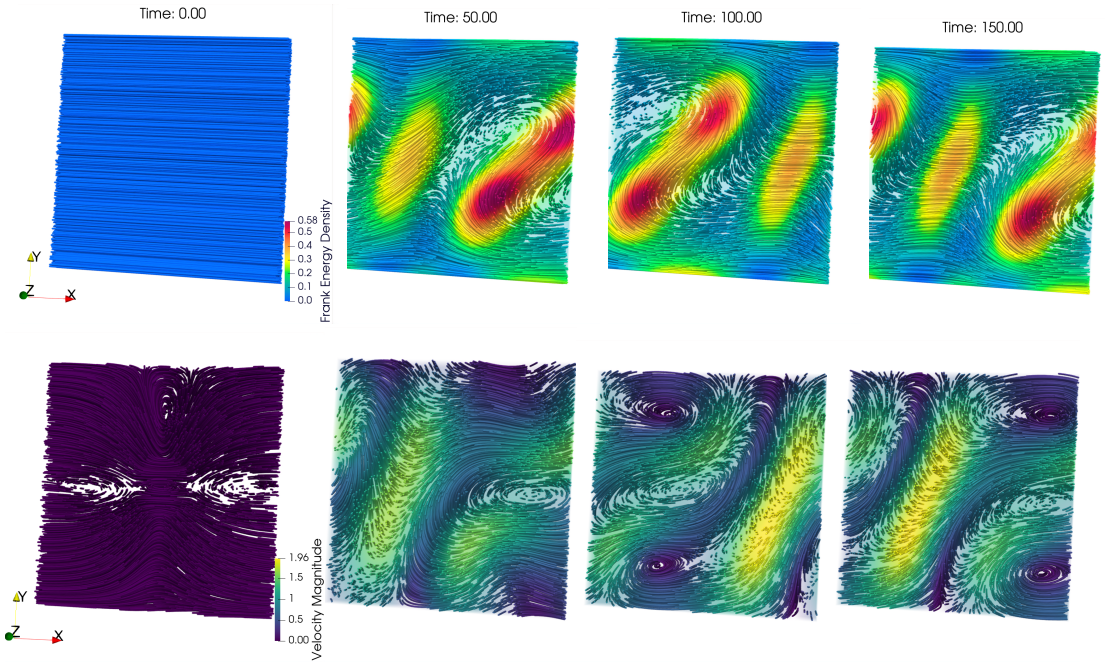



Figure 4.7: Transition phase diagram for different flow regimes. The solid lines plot the analytical expressions for the dimensionless critical activity $\Delta\tilde{\mu}_c$ vs. dimensionless flow-tumbling parameter $\tilde{\nu}$ for $\tilde{\gamma} = \tilde{\zeta} = 1$ and $\tilde{L} = 10$. The color of the ♦ symbols in the background grid indicates the maximum norm of the spontaneous flow velocity obtained by numerically solving the nonlinear equations at those parameters with a tolerance of $10^{-6}$. **(a)** Perpendicular polarity anchoring with extensile stress, Eq. (4.3). **(b)** Numerically obtained maximum flow velocity magnitude vs. activity $\Delta\tilde{\mu}$ for $\tilde{\nu} = -0.75$ (dashed vertical line in (a)) for spontaneous flow with perpendicular anchoring and extensile active stress. Note the broken Y-axis with different scales to accommodate for the sharp increase by 6 orders of magnitude around the critical activity $\Delta\tilde{\mu}_c$. **(c)** Parallel anchoring with contractile stress, Eq. (4.6); **(d)** parallel anchoring with extensile stress, Eq. (4.10);

parameter $\tilde{\nu}$ is plotted in Fig. 4.7d as a solid line. The numerical solutions for $\Delta\tilde{\mu} > \Delta\tilde{\mu}_c$ confirm the transition for different values of $\nu$, the maximum norm of the flow velocity is

shown as a color code in Fig. 4.7d.




(a) (b) (c) (d)

Figure 4.8: Traveling wave regime in a 3D active fluid depicted by polarity streamlines (top) and flow streamlines (bottom) over time. (**a**) Initial state at time 0 with 0.01 radian perturbation in the middle of the domain. (**b**) Transition to traveling wave state at time $t = 50$. (**c**) Appearance of tightly coupled counter-rotating vortices at $t = 150$. (**d**) Vortices traveling to the right $t = 150$.

## 4.3 Topological phase transition

In this section, we examine the behavior of extensile active fluids in regimes of increased activity. This is particularly relevant when these fluids have parallel anchoring, as shown in the previous section, that they promote out-of-plane fluctuations. In such conditions, perturbations can be substantial, rendering linear analysis inappropriate. Then, nonlinear numerical simulations become an essential tool for investigation. We further impose friction flow boundary condition on the top and bottom walls.

### 4.3.1 Traveling waves in active fluids

When the activity level surpasses a certain threshold, the active polar fluid enters a flow state known as traveling waves. This transition is marked by the emergence of pair of closeby, counter-rotating vortices. This phenomenon recalls the Berezinskii-Kosterlitz-Thouless (BKT) transition found in the XY model [174]. Our numerical study suggests that a similar transition in active fluids occurs at an activity level of $\Delta\mu \approx 4.35$.

Characteristics of the traveling waves regime include a wave that consistently moves in the positive or negative x-direction. It should be noted that the initial perturbation influences the direction of spontaneous symmetry breaking, thereby determining the resulting direction of the traveling wave. To illustrate this result, we present a time series of the polarity and velocity field streamlines in Figure 4.8, displaying the Frank energy density (Fig. 4.8 top), and the velocity magnitude (Fig. 4.8 bottom) at an activity level of $\Delta\mu = 6.0$. The snapshots are taken at times $t = 50$, $t = 100$, and $t = 150$.

At this activity level, the transitions between positively and negatively z-oriented polarization vectors are more pronounced, leading to a significant increase in the maximum Frank energy density. This results in the formation of polarity vortices near the velocity vortices.

In order to provide quantitative evidence of the traveling wave pattern, we compute the spatiotemporal correlation function $C_\varphi$. This involves calculating the correlation coefficient using the standard expression:

$$C_\theta(\delta\boldsymbol{x}, \delta t) = \frac{\mathrm{Cov}(\theta(\boldsymbol{x}, t), \theta(\boldsymbol{x} + \delta\boldsymbol{x}, t + \delta t))}{\sigma(\theta(\boldsymbol{x}, t))\sigma(\theta(\boldsymbol{x} + \delta\boldsymbol{x}, t + \delta t))} \tag{4.12}$$

In this equation, Cov represents the Covariance computed relative to the reference point at the center of the 3D box, $\boldsymbol{x} = \left(\frac{L_x}{2}, \frac{L_y}{2}, \frac{L_z}{2}\right)$, and $\sigma$ is the standard deviation. As can be seen in Figure 4.9, the correlation pattern translates along the x-axis over time, reinforcing the notion that both the polarization and velocity fields are traveling in the positive x-direction.

### 4.3.2 Intermittency and oscillatory flows

When the activity level rises to $\Delta\mu = 6.75$, we encounter a state of intermittent oscillatory waves. In this state, two waves that move in opposite directions periodically slow down before returning to the initial bending mode state. This pattern results in a characteristic frequency within the system, signifying a potential topological phase transition.

Strong time correlations between the flow and polarity fields, as well as the emergence of line defects, further hint at this transition. The intermittent oscillatory wave state bears similarities to the low-temperature state described in the BKT transition. As we further increase activity, we find that the system eventually becomes decorrelated and transitions into a chaotic state, interdispersed with periods of intermittent freezing, a behavior reported

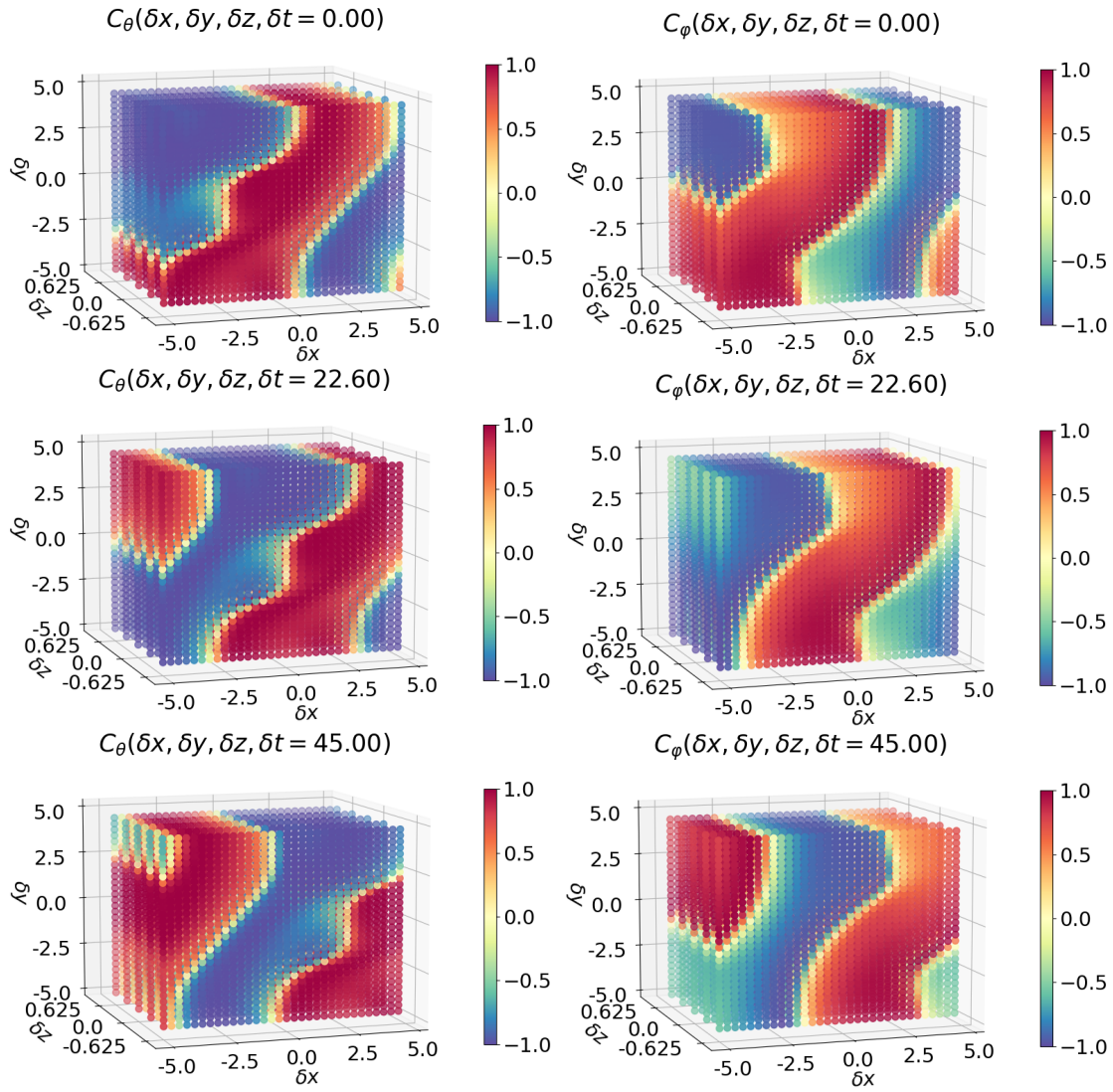
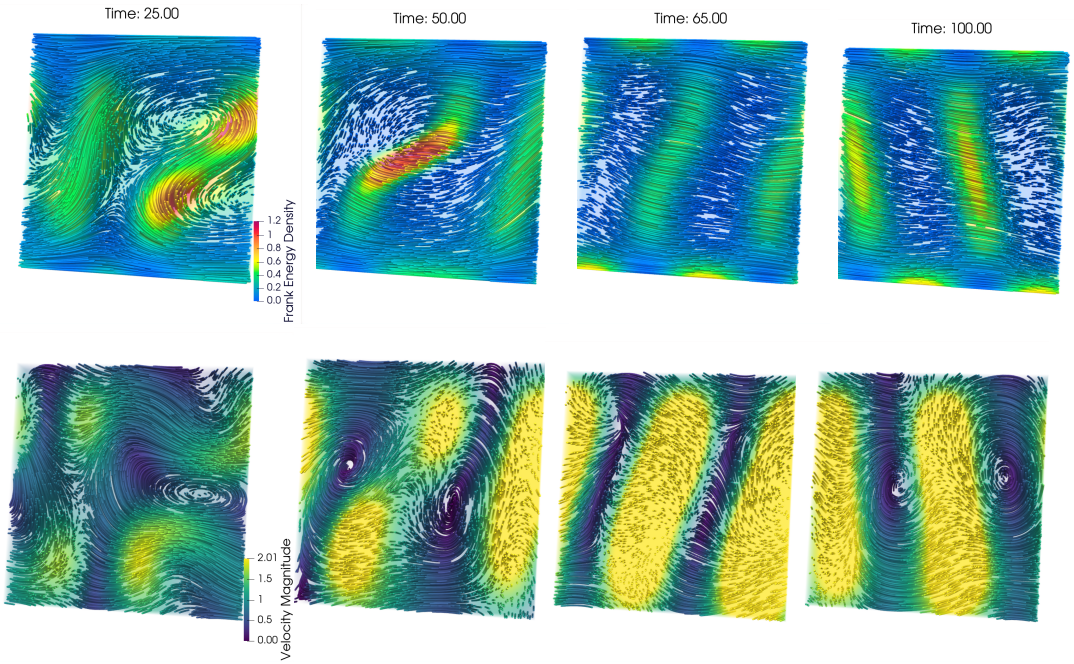



Figure 4.9: 3D Spatiotemporal correlation function for the traveling wave at $\Delta\mu = 6$ at different time windows. High correlation is observed for the traveling wave with counter-rotating vortices. Images adapted from [194].

in multiple dynamical systems [195] including spatiotemporal intermittency of the Ginzburg-Landau equation [196].




(a) (b) (c) (d)

Figure 4.10: Oscillations in 3D active fluids depicted by polarity streamlines (top) and flow streamlines (bottom) over time. (**a**) Transition state at time $t = 25$ with bending modes appearing. (**b**) Origin of traveling wave at time $t = 50$. (**c**) Waves and vortices traveling to the right-hand side at $t = 60$. (**d**) Waves and vortices traveling to the left hand side at $t = 100$.

### 4.3.3 Spatiotemporal chaos in active fluids

By substantially increasing the activity level to $\Delta\mu = 50$, we observe the onset of chaotic flow that rapidly becomes decorrelated as shown in Fig. 4.11, even over very short time frames. This behavior is visualized in Figure 4.12 showing the flow state at $t = 200$, revealing multiple vortex formations. This chaotic behavior of active fluids is often termed as Active Turbulence [197] in two dimensions. However, in three dimensions, we observe defect lines and loops that nucleate mainly from the domain boundary, dynamically appear as shown in Fig. 4.13, and may merge with other loops to disappear. The dynamics of the loops is topologically intricate and inherent to a three-dimensional active fluid. To further understand this chaotic behavior, we quantify the Lyapunov exponent over a broad range of activity levels.

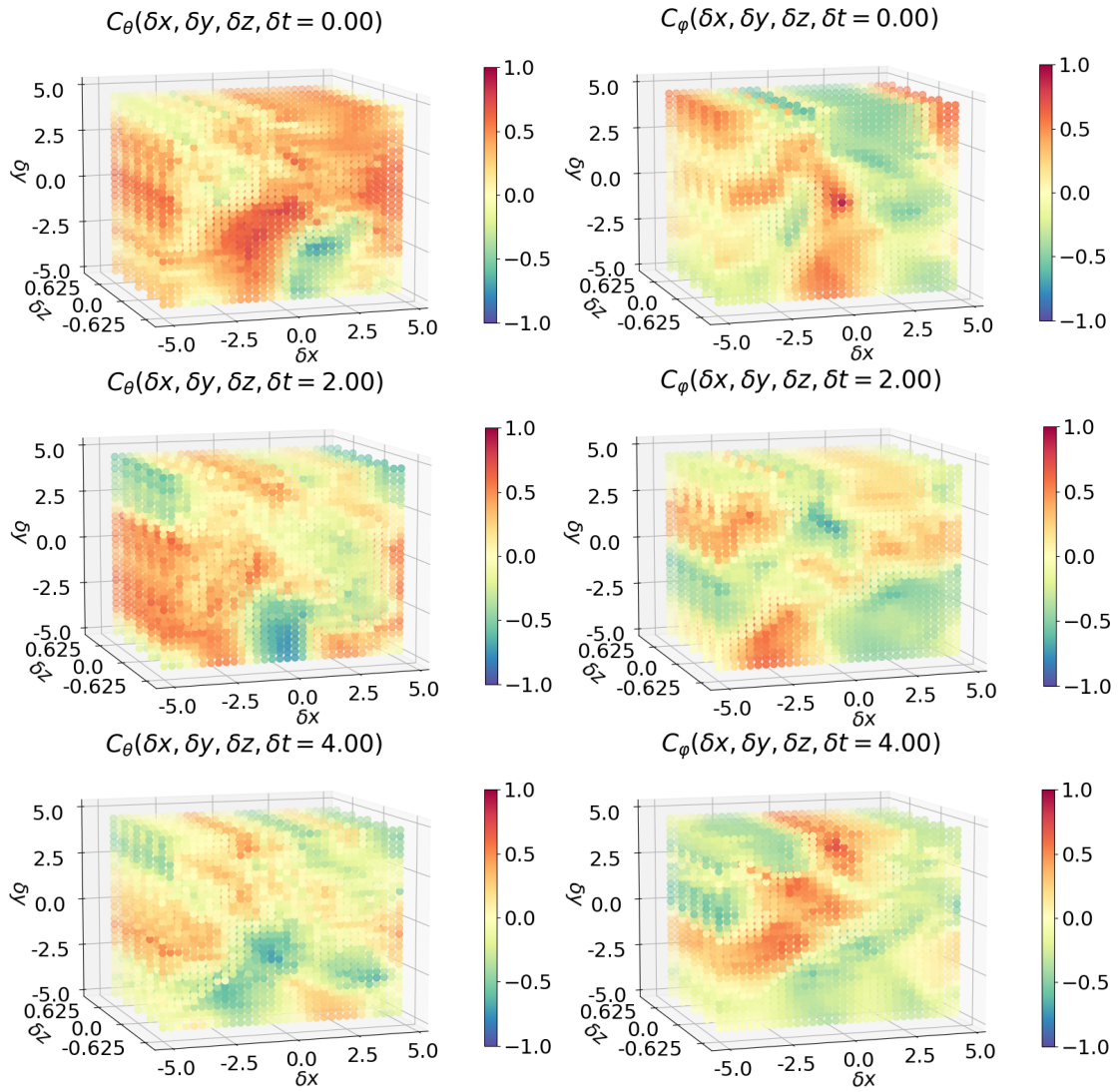



Figure 4.11: 3D Spatiotemporal correlation function for the chaotic state at $\Delta\mu = 50$ and different time windows. Little to no correlation is observed everywhere in the domain. Images adapted from [194].

### 4.3.4 Calculating the maximum Lyapunov exponent

The maximum Lyapunov exponent, denoted as $\lambda_1$, serves as an indicator of how a system reacts to small perturbations. The exponent is negative if a small perturbation dissipates and the system returns to its original, unperturbed state. Conversely, the exponent is positive if the perturbation grows. In cases where the perturbation remains steady, the maximum

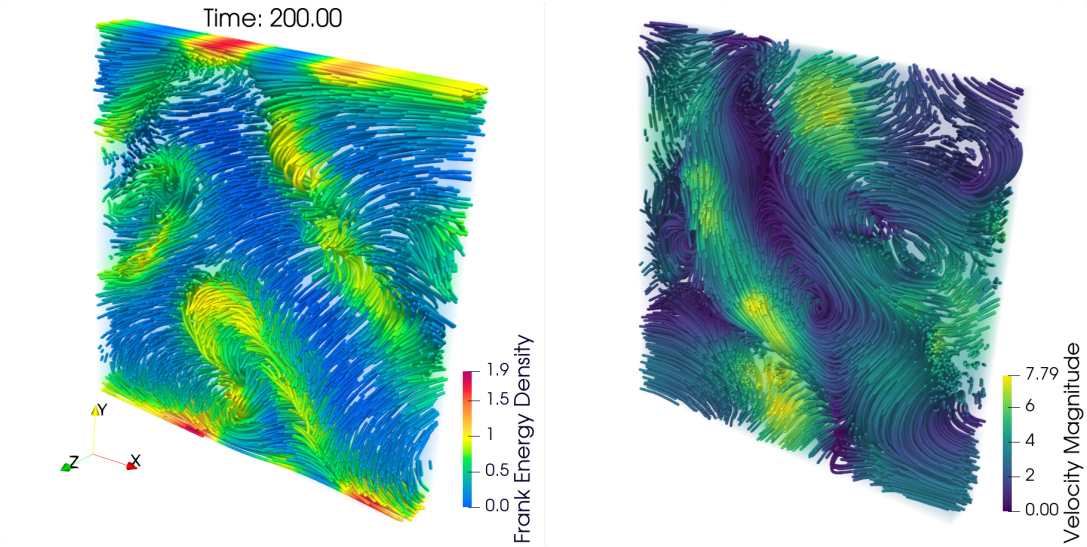



Figure 4.12: Simulation snapshot of polarity field streamlines (left) exhibiting line defects and velocity field streamlines (right) at $t = 200$ for the chaotic state at $\Delta\mu = 50$.

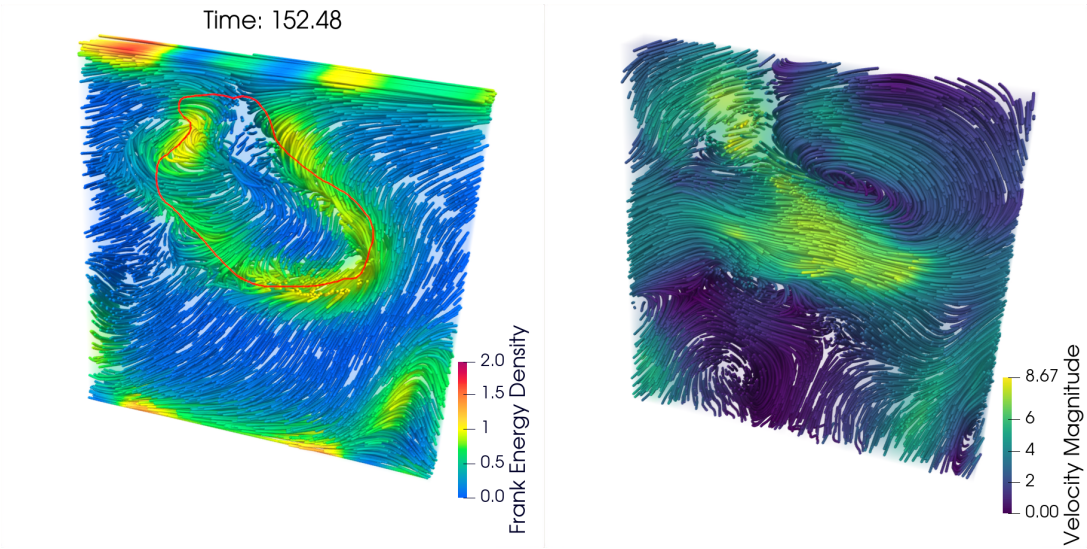



Figure 4.13: Simulation snapshot of polarity field streamlines (left) exhibiting a line defect loop (closed loop in red), and the corresponding velocity field streamlines (right) at $t = 152.48$ for $\Delta\mu = 50$.

Lyapunov exponent is zero. For the purpose of this study, we consider a perturbation vector $d_0$, that perturbs both angles $\theta$ and $\varphi$ by $\|d_0\| = 10^{-3}$ (where $\|\cdot\|$ represents the $L^2$-norm).

It's worth noting that "maximum" in this context refers not to the largest value among all simulation particles, but to the first of several Lyapunov exponents that describe the system's dynamic behavior. These exponents are also frequently referred to as Lyapunov characteristic exponents.

The computation of the maximum Lyapunov exponent follows the standard methodology proposed by Benettin et al. [198], as elucidated by Skokos [199] and implemented by Ramaswamy et al. [188]. The underlying concept entails initiating two concurrent simulations, $S_1$ and $S_2$, using identical models and simulation parameters, with the exception that the center particle's polarization in simulation $S_2$ is further perturbed by $\|d_0\|$.

Following the evolution of both simulations for a specified time window, $T_L$, the difference, $\|d_i\|$, (in angles $\theta$ and $\varphi$) between the two simulations is calculated. The maximum Lyapunov exponent for this particular time window is then given by:

$$X_1(T_L) = \frac{1}{T_L} \ln \frac{\|d_i\|}{\|d_0\|}. \tag{4.13}$$

Subsequently, the difference $\|d_i\|$ between the two simulations is reset to the fixed norm $|d_0|$. This procedure is repeated for a predetermined number of time windows, denoted as $k$. The final value of the maximum Lyapunov exponent is the long-time average of $X_1$, calculated as:

$$\lambda_1 = \lim_{k\to\infty} X_1(kT_L) = \lim_{k\to\infty} \frac{1}{kT_L} \sum_{i=1}^{k} \ln \frac{\|d_i\|}{\|d_0\|}. \tag{4.14}$$

**Algorithm 4.1** Maximum Lyapunov Exponent Calculation

1: Initialize particles in both simulations $S_1$ and $S_2$
2: Perturb the central particle of $S_2$ by $|d_0|$
3: Define $k = tf/T_L$
4: Set $i = 0$
5: **while** $i =< k$ **do**
6: Evolve both $S_1$ and $S_2$ from time $t = iT_L$ to $t = (i + 1)T_L$
7: Calculate the current value of $|d_i|$
8: Compute $X_1$ as $X_1 = \ln |d_i|/|d_0|$
9: Reset $|d_i|$ to $|d_0|$
10: Increment $i$ by 1
11: Return $\lambda_1 = X_1(kT_L)$

The algorithm for calculating the Maximum Lyapunov exponent is provided in Algorithm 4.1. The chosen time window is $T_L = 0.5$, equivalent to 50 steps of time integration, and we

consider a total of $k = 800$ time windows. The average of Eq. (4.14) is derived over the final 300 time windows. It is important to note that for the simulations used to determine the Maximum Lyapunov exponent, the steady state condition $\epsilon_{\text{steady}}$ is not used. We visualize the numerically computed Lyapunov exponents in Fig. 4.14

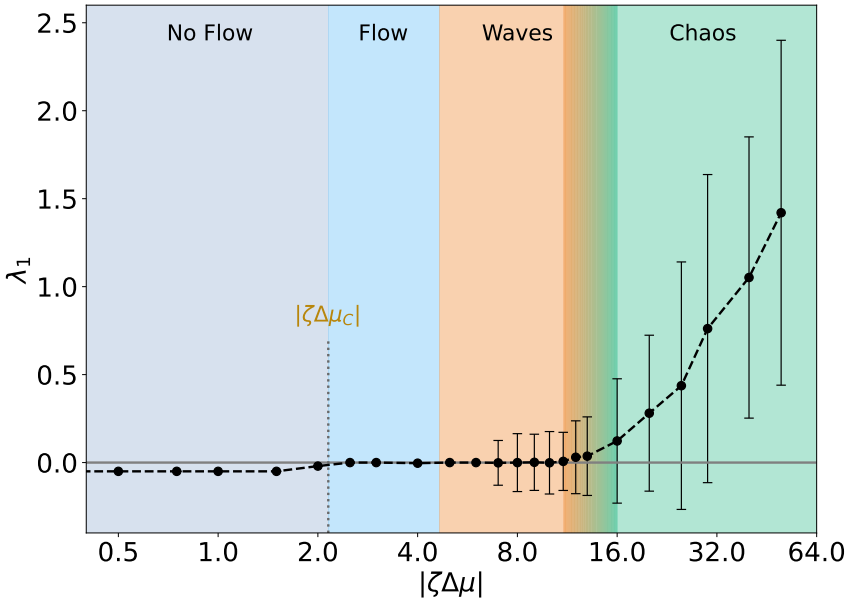



Figure 4.14: Numerically computed maximum Lyapunov Exonent $\lambda_1$ for various activity levels (dots: mean; error bars: standard deviation) as a function of the activity level. For increasing activity, the system transits from no flow, to flow, to traveling vortices, and eventually chaos. The increasing standard deviation in the waves regime is caused by polarity oscillations.

## 4.4 Conclusion

We used the new simulation algorithm to unveil a critical spontaneous flow transition in three-dimensional (3D) active fluids and described its behavior under various boundary conditions and sign of the active stress (contractile vs. extensile)

Under perpendicular anchoring, the spontaneous flow transition notably deviates from the two-dimensional (2D) case, as it generates a shear flow along both X and Z axes. The occurrence of such a flow leads to a peculiar phenomenon of out-of-plane bending of polarity, giving rise to a spontaneous 3D flow transition invariant along the X and Z directions.

It is important to highlight that this transition adds a significant level of complexity to the already intricate dynamics of active fluids. This complexity arises from the out-of-

plane perturbations, which add a third dimension to the dynamics when compared to the conventional 2D scenario. Such out-of-plane effects could have far-reaching implications in many areas of research where active fluids play a central role, including understanding the movement of living cells or the development of novel materials.

In the situation where parallel anchoring is enforced at the wall, the transition behaves differently. Then, contractile active stress counteracts out-of-plane perturbations and leads to an invariant extension of the 2D spontaneous flow transition. This observation further emphasizes the role of boundary conditions in selecting the dynamic behavior of active fluids.

Intriguingly, we also identified a unique out-of-plane "wrinkling" phenomenon under extensile active stress, a behavior which is absent in 2D domains with rigid boundaries. This "wrinkling" could offer explanation to multiple experimentally observed instabilities in 3D microtubule assays with an extensile active stress [9, 20, 43], potentially elucidating new complex system parameters.

Beyond this, the study of active fluids under escalating activity revealed a diverse array of intricate states, transitioning from traveling waves to intermittency, and ultimately to spatiotemporal chaos. These states mirror the patterns found in the well-established Berezinskii–Kosterlitz–Thouless (BKT) phase transitions, where two-dimensional systems undergo phase transitions caused by topological defects or vortices. The BKT transition has found prominent applications in understanding various systems in condensed matter physics. Applying the principles of the BKT transition to our study has provided valuable insights such as an existence of a topological phase transition in active matter systems. Further, we observe three dimensional line defect loops that are spontaneously created and destroyed. The dynamics of the defect loops is topologically intricate and may exhibit complex braiding behavior that needs further exploration. It is indeed fascinating to observe that these transitions also manifest in active fluids, which offers a richer understanding of their behavior across activity levels. These findings not only extend the reach of BKT transitions to active matter, but also significantly contribute to the growing body of knowledge in the realm of active fluid dynamics.

In summary, this chapter demonstrates the potential of our novel numerical solver to study the intricate dynamics and transitions inherent to active fluids in different conditions and across varying activity levels. These insights broaden our understanding of active fluid behavior and provide a valuable foundation for both theoretical advancements and practical applications in related fields. Future work will include a detailed study of the dynamics of the defect loops, contributing further to a topological understanding of active fluids.

# Chapter 5

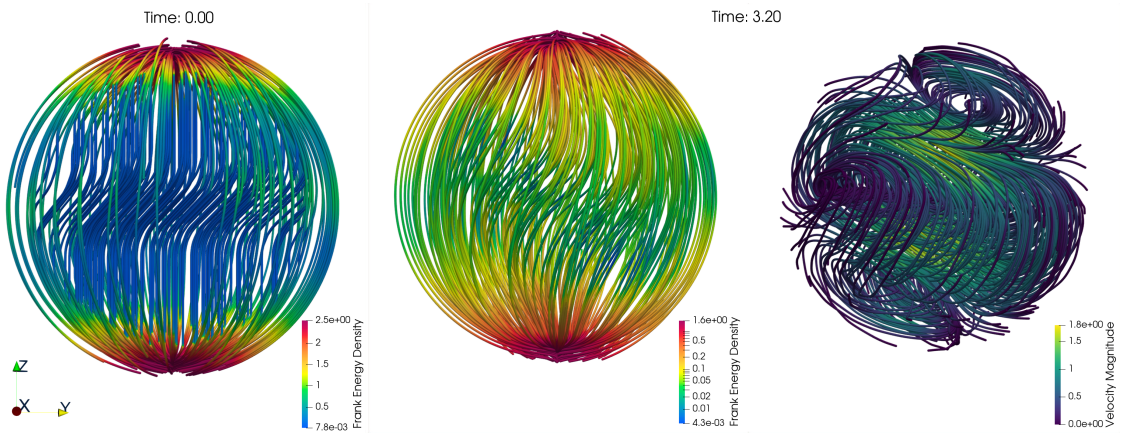
# Active hydrodynamics in 3D complex geometries

In the previous chapter, we studied active hydrodynamics in a rectangular Cartesian geometry. In this chapter, we simulate active hydrodynamics in three dimensional non-Cartesian domains. This results in uncovering rich behavior of active fluids when these systems are confined within curved domains, starting with spherical droplets.

Significant advancements have recently been made for active matter within spheroidal confinement, modeling the eukaryotic nucleus [200]. The exploration of the large-scale organization of the genome within the cell nucleus underscored the influence of active mechanical processes and hydrodynamic interactions on chromatin compartmentalization [201]. Simultaneously, studies using multiscale models of long flexible polymers in active fluids, akin to chromatin fibers in nucleoplasm, have elucidated the emergence of coherent motions and structured conformations driven by hydrodynamic and alignment interactions [202]. These underscore the critical interplay between active and passive components in biological systems under confinement with curvature.

Our study is motivated by recent experiments as shown in Fig. 1.4e where microtubule assays were assembled with filamentous FD virus (F-specific filamentous phages possessing the fertility (F) factor) in water droplets [57]. FD virus provides nematic elasticity to the microtubules, and addition of active Kinesin-Streptavidin cluster crosslinks the microtubules and allows Kinesin motors to generate extensile active stress. These molecules make the system an active liquid crystal with nematic/polar ordering that is ideal for testing our vectorial active hydrodynamics model introduced in Eq. (3.1). Here, we find that a curved domain significantly increases the critical active potential for the spontaneous flow transition. We then explore the behavior of the same model in a 3D annuli and "peanut" shaped geometry, where we show that boundary effects become even more significant for the resulting flow behavior, and show how they can be harnessed for coherent and chiral flows.

Motivated by experiments on cytoskeletal actin and myosin extracts obtained form




(a) (b)

Figure 5.1: Spontaneous flow transition in an extensile active fluid confined to a 3D ball. **(a)** Polarity field streamlines of the initial state with Frank energy density as color. **(b)** Spontaneous flow steady-state at time $t = 3.2$ with bent polarity field streamlines (left) and opposite shearing flow velocity was found with circulation (right), and velocity magnitude as color.

*Xenopus* egg extract, we further study a density-based active hydrodynamic model in a complex spherical shell domain. The experimental system exhibits spontaneous flow when confined within a droplet leading to formation of an opaque inclusion at the center of the droplet as shown in Fig. 5.5a. We perform simulations of laser ablation to elucidate mechanisms by which actin monomers polymerize and create spontaneous flows. We further demonstrate simulations with moving boundaries and the computation of restoring forces that are experimentally observed with the help of magnetic beads and tweezers, but are challenging to estimate from the experiments.

## 5.1 Active nematic balls in 3D

When microtubule assays are assembled with filamentous FD virus and active Kinesin motors in water droplets, they exhibit spontaneous flows under certain conditions [57]. Experimentally, the Duclos lab observed a behavior is similar as our theoretical predictions in 3D channels [9, 43, 171]. The active fluid is unstable above a critical active stress or, equivalently, above a critical radius or length scale. This suggest that there is an underlying hydrodynamic instability inherent to the system. Hence, we perform numerical simulations in a 3D ball of the previously described model Eqs. (3.1). We discretize the 3D ball with meshless particles in an Eulerian frame of reference. We use the algorithm as described

in section 3.4 to study the time evolution of the polarity and velocity fields. The system necessitates imposing boundary conditions on the curved surface. Experimentally, it is unclear what happens at the boundary in the flow state. However, in the no-flow state, many droplets exhibit nematic order along the surface with two defects located diametrically opposite [57]. This configuration is the minimum-energy configuration due to the Poincare-Hopf theorem. Hence, we use the first mode of the vector spherical harmonics as initial boundary condition. We first enforce Dirichlet boundary conditions where the polarity with two opposite facing defects remains fixed. This is analogous to surface anchoring boundary conditions. For comparison, we also perform the simulations with Neumann boundary conditions for the polarity and find qualitatively similar behavior in the bulk dynamics. For both cases, we enforce the natural no-slip boundary conditions for the flow.

### 5.1.1 Dirichlet boundary conditions

We fix the boundary polarity field for modes $l = 1$ and $m = 0$, with vector spherical harmonics as described in section 3.5. We then initialize the polarity field at $t = 0$ in the bulk of the 3D ball to be pointing upwards in the $Z$ direction, apart from a perturbation of $\frac{\pi}{4}$ radians in both $X$ and $Y$ directions in the plane $Z = 0$ as shown in Fig. 5.1a. Simulating the dynamics with our meshfree solver for the parameters $\eta = 1, \gamma = 1, \zeta = 1, \lambda = 0, K_s = K_b = K_t = K = 1$, and $\nu = -1$. We find that the perturbations relax to a homogeneous aligned state and there is no flow below a critical radius or activity. However, for this extensile regime, above the critical activity or above a critical radius as shown in Fig. 5.2a-b, we find a spontaneous flow transition with opposite shearing flows, similar to the flows described for the bend instability case in the previous chapter. We find a flow steady-state that exhibits a wrinkling instability as shown in Fig. 5.1b with opposite shearing flows. This is qualitatively in agreement with experimental observations, which also suggest a nematic flow instability above critical motor concentration or droplet size.

Further increasing the activity or the droplet size above the critical point, we first observe a regime of oscillatory periodic flows, and eventually, spatio-temporal chaos. The total Frank energy and the mean flow magnitude for various radii values are visualized in Fig. 5.2c-d.

### 5.1.2 Neumann boundary Conditions

For comparison, we also run the simulations with Neumann boundary conditions that set the derivative of the polarity field normal to the boundary to 0. This adds an extra layer of complexity for imposing such boundary conditions during time integration. We take advantage of our meshfree discretization to impose these conditions. This is done by creating a thin ghost layer of particles that is outside the domain in the normal direction to the surface with a fixed spacing. Using the correspondence between the source and the ghost particles, a simple copy operation of the polarity fiield from source to ghost produces

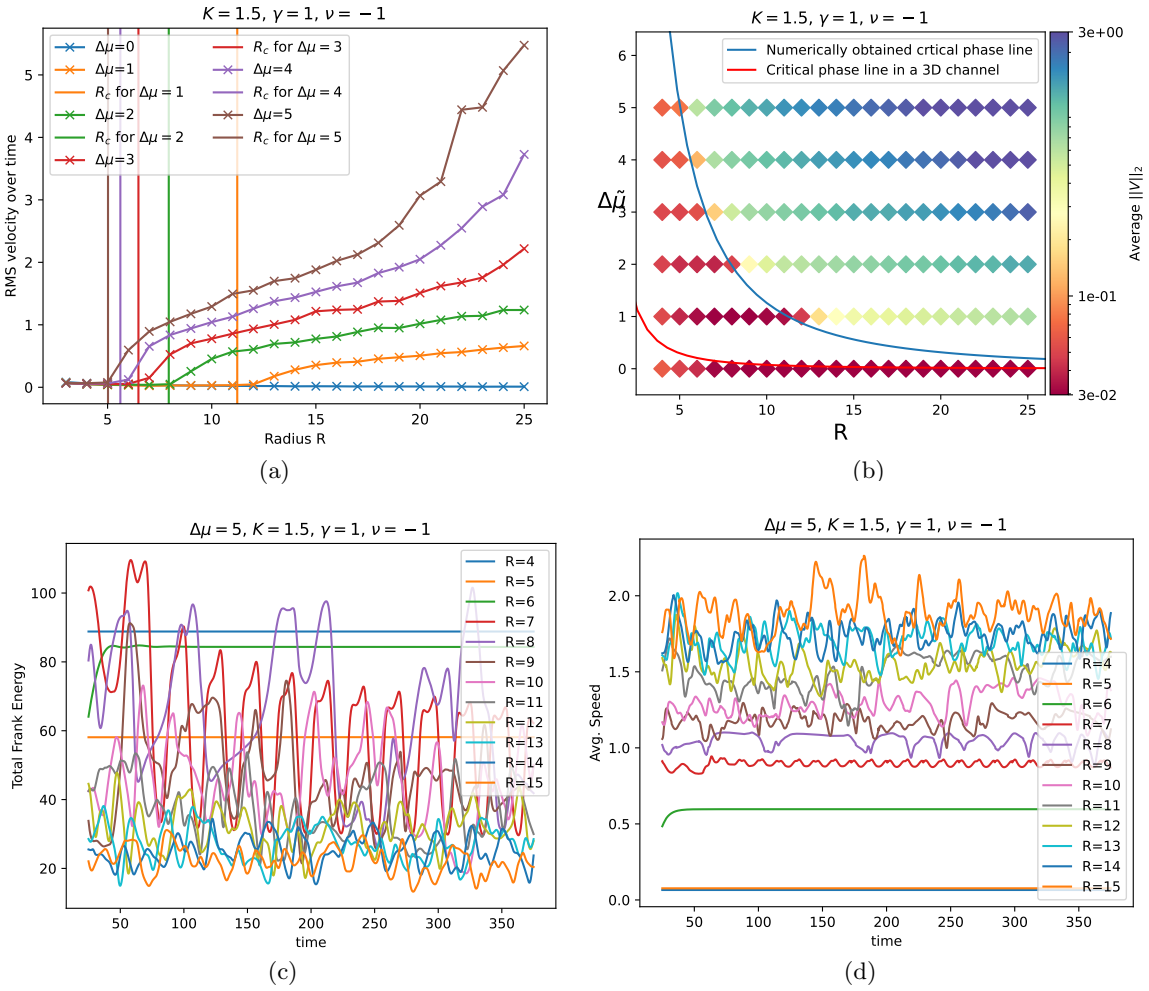
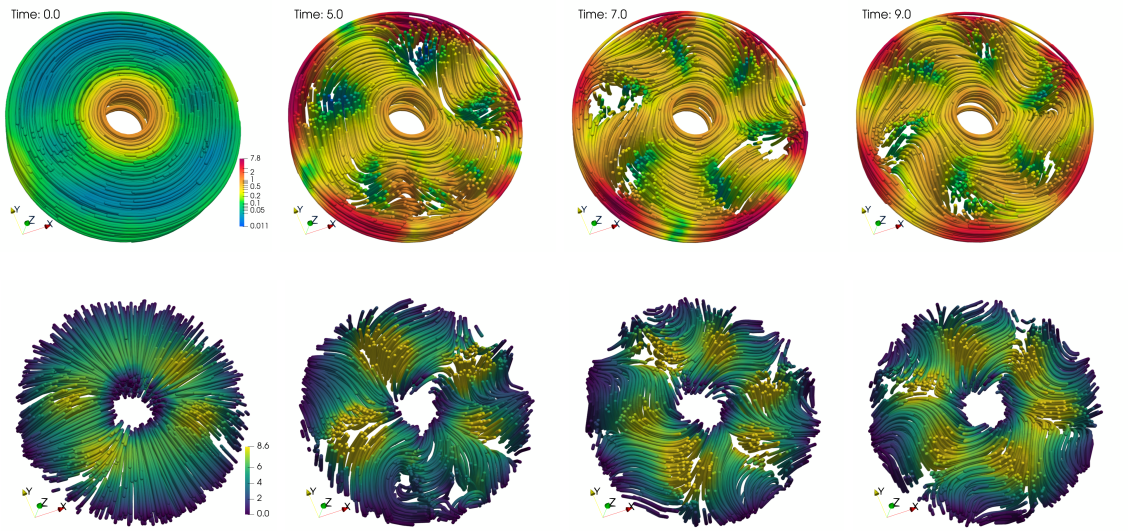



Figure 5.2: Numerical simulation of active polar fluid in a 3D ball. **(a)** Root-mean-square (RMS) velocity vs radius of the 3D ball at $t = 100$. **(b)** Spontaneous flow phase diagram for different activity levels and radius. The blue line indicates numerically obtained critical phase line that is much higher for a 3D ball than the analytical prediction Eq. (4.10) in a 3D channel. The color of the ♦ symbols in the background grid indicates the maximum norm of the spontaneous flow velocity at $t = 100$. **(c)** Time series plot of the total Frank energy for different radius $R$. **(d)** Time series plot of the average RMS flow velocity for different radius $R$.

the boundary polarity field with no gradient in the normal direction.

We do not observe any different phenomenology from the Dirichlet case, except that the boundary polarity can now relax to a fully homogeneous state with no defects. In this




(a) (b) (c) (d)

Figure 5.3: Coherent motion of an active fluid in an annular domain depicted by polarity streamlines (top) and flow streamlines (bottom). (**a**) Initial state at time 0. (**b**) Bend instability onset at time $t = 5$. (**c**) Rotation of the bent state at time $t = 7$. (**d**) Rotating state at time $t = 9$.

state, the polarity field points outside the surface in the radial direction. In the increased activity regime, we find that the polarity field exhibits dynamical hairy structures at the boundary, while exhibiting chaotic flows.

## 5.2 Coherent motion of active nematics in 3D annuli

Moving to another non-Cartesian geometry, we test our algorithm as described in chapter 3 in a 3D annulus. The behavior of active matter in 2D annular geometries has been extensively studied both experimentally [203–206] and numerically [207–209]. The main focus of these studies was on confinement-induced transitions to active turbulence. In 3D, an experimental study reported coherent motion of microtubule assays confined to an annulus [210], but we are not aware of any numerical simulations of this system in 3D.

We here solve Eqs. (3.1) of active polar hydrodynamics in a 3D annulus of inner radius 1 (dimensionless units), outer radius of 5, and thickness of 2.75 in the periodic Z-direction. The annulus is centered at (0, 0, 1.375). At the inner and outer ring surfaces, the polarity field is anchored tangentially and parallel to the XY-plane. For the velocity field, physically realistic no-slip boundary conditions are imposed. We use the same model and simulation parameters $\eta = 1, \gamma = 1, \zeta = 1, \lambda = 1, K_s = K_b = K_t = 1$, and $\nu = -1$, $\Delta\mu = 60$. We

perform the simulation in the Eulerian frame of reference using the method of images to impose the boundary conditions, which would be harder to do in the Lagrangian reference frame in this case. At $t = 0$, polarity is homogeneously aligned tangentially and parallel to the $xy$-plane with small random perturbations of 0.01 radians in the $z$ direction for $1 < |x| < 2.5$, $-1 < y < 1$, and all $z$ values in the domain (Fig. 5.3a). We observe that the polarity develops a circular variant of the bend instability at $t \approx 5$ (Fig. 5.3b). This bent state develops a six-fold angular symmetry and exhibits coherent flows rotating it clockwise (Fig. 5.3c-d). This confirms the experimentally observed [210] emergence of coherent angular motion in 3D annular domains and suggests that extensile active materials can use the bend instability to sustain a bent mode for generating coherent motion. In comparison to 3D channels, similar activity levels lead to chaotic flows. This suggests geometry can play an important role in controlling the behavior of an active fluid.

## 5.3 Chiral vortices in asymmetric 3D geometries

Next, we demonstrate the robustness and versatility of the present simulation algorithm in more challenging 3D geometries. We create two different geometries using Blender [211] that resemble shapes of dividing biological cells. The first geometry is a rotationally and left-right symmetric "peanut" shape, discretized with 1952 boundary particles given by the vertices of Blender's triangular surface mesh and 10,627 bulk particles placed in a regular Cartesian grid with spacing $h = 0.1$. The second geometry is an asymmetric geometry created by manually indenting the "peanut" mesh at arbitrary locations, different for the left and right sides. It has the same number of boundary particles and 10 898 bulk particles with the same $h = 0.1$. At the boundary, polarity is anchored to be perpendicular to the boundary, and the velocity is zero The initial condition consists of a random polarity field with polarity at each bulk particle sampled uniformly randomly on the unit sphere. The initial velocity field is zero everywhere. We use the same model and simulations parameters as in Sec. 4.2.2, except for $\nu = 0$ and $\epsilon_v = 5 \cdot 10^{-4}$. For both geometries, we test the two cases of $\Delta\mu = 0.0$ and $\Delta\mu = 20.0$. All simulations run until $t_f = 10.0$. Due to the difficulty of applying the method of images in complex geometries, we use the Eulerian frame of reference for this simulation.

Without activity, the polarity field aligns with the boundary condition, and the velocity decays to zero everywhere in the domain. This is as expected for a purely dissipative system and confirms the physical plausibility of the simulation. At $\Delta\mu = 20.0$, a steady state with non-zero flow develops in both the symmetric and the asymmetric geometries (Fig. 5.4). The polarity and velocity fields reflect the symmetry properties of the domain. In the symmetric shape (Fig. 5.4a–c), the fields are approximately left-right symmetric, but not exactly, since we start from a random initial polarity field. Over time (panels from left to right), the polarity field develops two stable asters in either body of the "peanut", and the velocity field becomes chiral with multiple stable vortices stemming from the antisymmetric

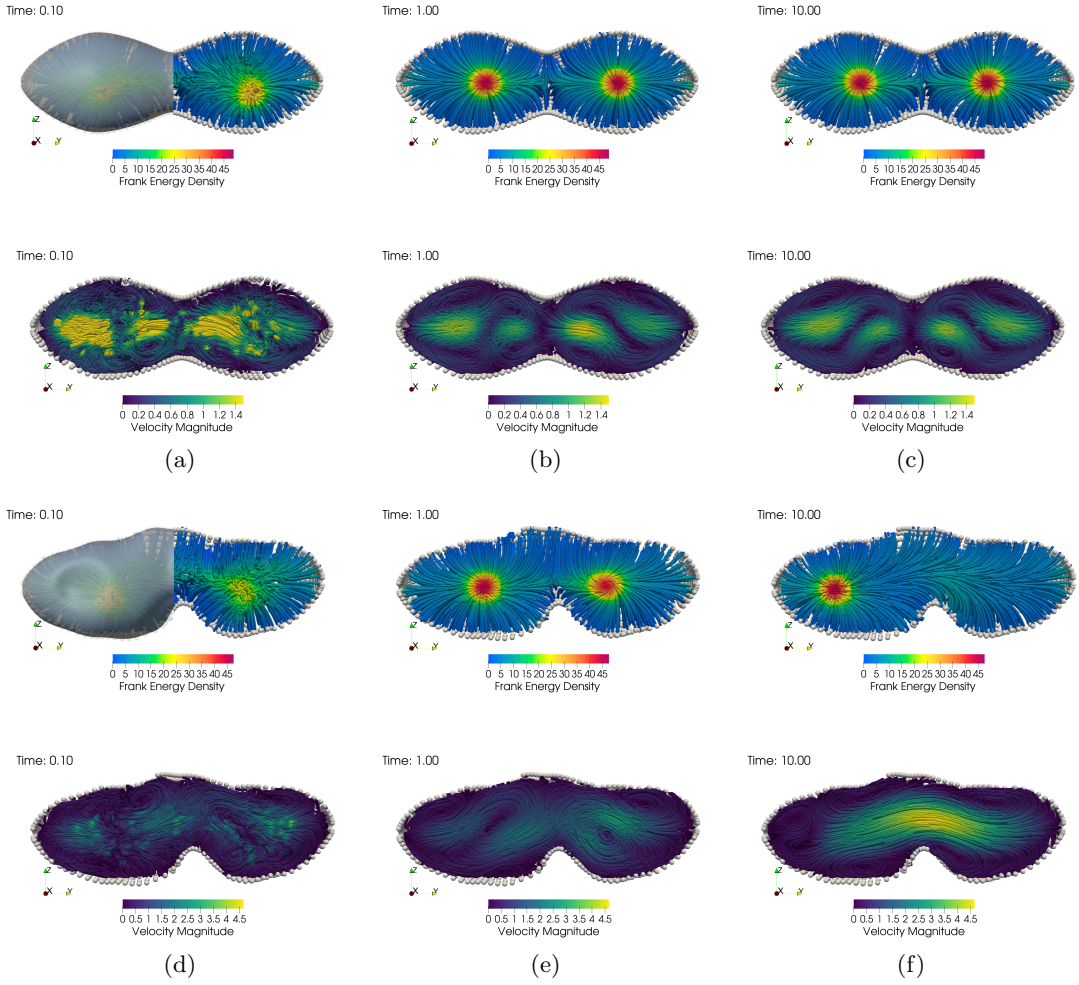
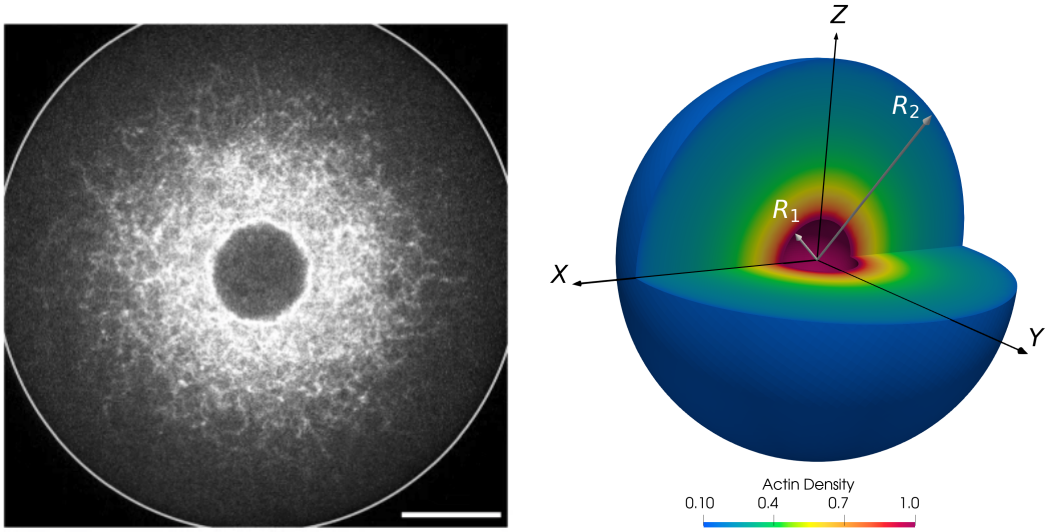



Figure 5.4: Developing polarity (top, color bar: Frank free-energy density) and velocity (bottom, color bar: velocity magnitude) streamlines of an extensile active polar fluid in a symmetric (**a**–**c**) and an asymmetric (**d**–**f**) peanut-like 3D geometry. From left to right, we show the flow fields for increasing time ($t = 0.1$, 1.0, 10.0) as indicated in the panels. For better visualization, the shapes are cut open along their axial mid-planes. Gray spheres show boundary particles. A surface rendering of one half of either shape is overlaid to the first frame as a gray, semitransparent surface.




(a) (b)

Figure 5.5: **(a)** Spontaneous flow in cytoskeletal extracts of *Xenopus* egg. Image adapted from [58]. **(b)** Steady-state of the actin density in the 3D simulation.

part of the stress tensor. In the asymmetric shape (Fig. 5.4d–f), a similar pattern initially forms at $t \approx 1.0$, but later one aster "eats" the other, and a strong steady-state streaming velocity is induced through the neck between the two sides of the "peanut". Taken together, these simulations not only showcase the application of the present simulation framework to arbitrary 3D geometries, but predict the emergence of chiral flows in axially symmetric shapes and geometry-induced left-right symmetry breaking in asymmetric domain shapes.

Up to this point, we have only adjusted the domain's geometry while maintaining the model equations unchanged. We now introduce an example that requires modifications to both the model equations and domain geometry. This showcases the versatility our numerical and software framework.

## 5.4 Hydrodynamics of cytoskeletal actin flows

Demonstrating the developed expression-system further, we explore the hydrodynamics of cytoskeletal actin flows, illustrating its capability to handle intricate cellular dynamics. We explore the dynamic behavior of cytoskeletal extracts from *Xenopus* eggs, which consist of actin and other proteins. These extracts exhibit intriguing active hydrodynamic characteristics when confined to water droplets submerged in oil, including the spontaneous formation of an inclusion at the droplet center. The inclusion results from the self-driven

flows generated by actomyosin polymers and motor proteins. We simulate this system to showcase the application of the present framework to biological instances of active matter, with a particular focus on the hydrodynamics of cytoskeletal actin flows.

We study a spherical water-based droplet with radius $R_2$, containing a contractile acto-myosin network and encased in oil. The inclusion within the droplet, made up of displaced debris, is represented as a rigid sphere with radius $R_1 < R_2$ (see to Fig. 5.5b). We model the contractile acto-myosin network as a one-component active viscous fluid made of polymerized actin, characterized by mass density $\rho$ and velocity $\mathbf{v}$.

### 5.4.1 Nonlinear isotropic model with threshold

#### Actin density continuity equation

The polymerized actin density $\rho$ follows a continuity equation, since actin can be advected by the flow and may undergo polymerization and depolymerization:

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = f(\rho)\,, \tag{5.1}$$

Here, $f(\rho)$ is the actin turnover, dependent on the concentration of polymerized actin, as further discussed below.

#### Force balance and constitutive equations

The force balance in the actin gel is given by:

$$\mathrm{div}\sigma = -\mathbf{f}^{\mathrm{ext}}\,, \tag{5.2}$$

where $\sigma$ is the stress tensor of the actin gel. In this one-fluid model, the external force arises from the friction between the actin gel and the fluid surrounding it:

$$\mathbf{f}^{\mathrm{ext}} = -\gamma(\rho)\mathbf{v}\,, \tag{5.3}$$

where $\gamma(\rho)$ is a density-dependent friction coefficient.

Considering the experimental evidence that the polymerized actin network is isotropic, we propose the following constitutive equation for the actin gel stress:

$$\sigma_{\alpha\beta} = 2\eta(\rho)\tilde{v}_{\alpha\beta} + \left[\eta_{\mathrm{b}}(\rho)\partial_\gamma v_\gamma) - P(\rho)\right]\delta_{\alpha\beta}\,, \tag{5.4}$$

where Greek indices denote Cartesian coordinates and where an implicit summation over repeated indices is implied (Einstein notation). We have introduced the shear rate tensor $\tilde{v}_{\alpha\beta} = (\partial_\alpha v_\beta + \partial_\beta v_\alpha)/2 - (\partial_\gamma v_\gamma/3)\delta_{\alpha\beta}$. We have also introduced the active isotropic stress $-P\delta_{\alpha\beta}$, and the shear and bulk viscosities $\eta$ and $\eta_{\mathrm{b}}$ respectively, which may all depend on the local gel density $\rho$. We next discuss this density-dependent behavior.

**Threshold density**

The actin network's properties are highly dependent on its density. From recent observations by Jianguo Zhao at Christoph lab, Duke University, North Carolina, USA and at Jülicher group, MPI-PKS, Dresden, Germany, a threshold density $\rho_c$ is observed below which there is net production of polymerized actin ($f(\rho < \rho_c) > 0$), and above which actin is, on average, depolymerized ($f(\rho > \rho_c) < 0$). This behavior is modeled using a sigmoid-like function for the actin turnover:

$$f(\rho) = k_0 + k_0' \theta_w(\rho - \rho_c) + k_1 \rho \,, \tag{5.5}$$

where we have introduced the nonlinear function $\theta_w(\rho) = [1 + \tanh(\rho/w)]/2$ that blends between the two linear turn-over regimes, with $w$ denoting the width of the transition region. The corresponding fit to the experimental data that shows excellent agreement will be presented in a future publication [212]. Based on this experimental evidence, we postulate that the mechanical properties of the actin network also depends on the threshold density, and that below this threshold density the actin network is too sparse to exert active forces and essentially behaves as a passive viscous fluid, while above it, the network is sufficiently dense to exert contractile stresses that can drive fluid flows. We therefore consider the following density dependence of the active stress in Eq, (5.4):

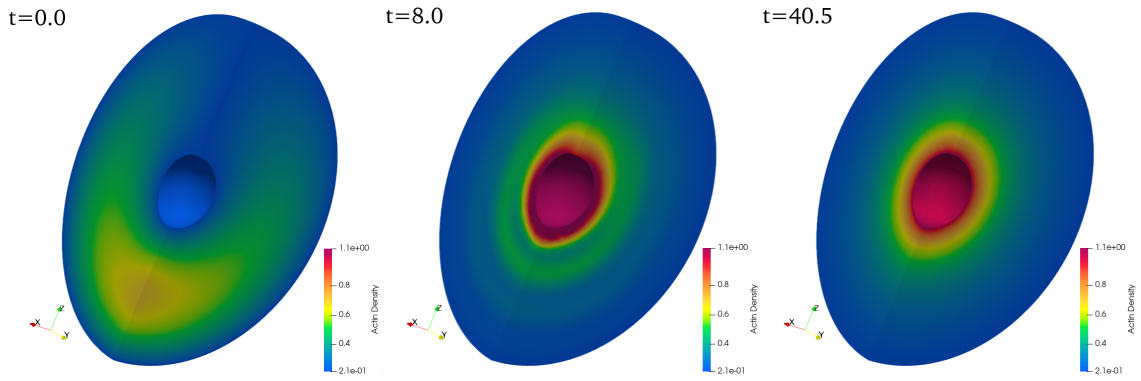
$$P(\rho) = P_0 + \left[P_0' + P_1' \rho\right] \theta_w(\rho - \rho_c) \,, \tag{5.6}$$

while we assume that the friction $\gamma$ and the shear and bulk viscosities $\eta, \eta_b$ are density independent. Note that in contrast with previous work [58], we neither require density dependent viscosities, nor any special relationship between the viscosities and the isotropic active stress in order to obtain good fits of the experimental data. In particular, we are able to obtain this characteristic linear velocity profile ($v \simeq a + br$) for $R_1 < r \lesssim 0.8 R_2$ that was also reported in Ref. [58].

To validate a full 3D simulation and to fit the experimental data, we consider a spherically symmetric one dimensional model (see Appendix C.1). We find that both the one dimensional, and the three dimensional solver converge to the same density and velocity profiles.

### 5.4.2 Boundary conditions for 3D non-linear simulations

To solve the system of nonlinear differential equations (5.1)-(5.6), we need to specify three boundary conditions, which have to be imposed at the boundary with the outer droplet and at the boundary with inclusion. We describe the boundary conditions in the spherical coordinates for easier readability.




(a) (b) (c)

Figure 5.6: Temporal progression of the actin density distribution in response to an uneven initial actin concentration (a quarter-section of the spherical shell is depicted). **(a)** Initial state of the actin density at $t = 0$. **(b)** Actin density at $t = 8.0$. **(c)** Steady-state actin density at $t = 40.5$.

### Actin concentration

The first boundary condition we impose is a fixed actin concentration at the surface $\mathcal{S}_1$ of the inclusion:

$$\rho(\mathbf{r} \in \mathcal{S}_1, t) = \rho_1 \,. \tag{5.7}$$

However, since the flow resulting from actin contraction is directed inwards (towards the inclusion), imposing the boundary condition at the inclusion can be challenging. Depending on the simulation method used (see Appendix. C for details), we have alternatively imposed the actin density at the outer surface $\mathcal{S}_2$, $\rho(\mathbf{r} \in \mathcal{S}_2, t) = \rho_2$.

In the spherically-symmetric case, the inclusion is centred at the origin and the boundary conditions reads:

$$\rho(r = R_1, t) = \rho_1 \,, \tag{5.8}$$

or $\rho(r = R_2, t) = \rho_2$ when the boundary condition is imposed at the outer surface.

### Actin velocity

We impose two boundary conditions for the actin velocity $\mathbf{v}$, one at the surface of the inclusion and one at the outer surface of the droplet. We have considered both fixed velocity and stress-free boundary conditions.

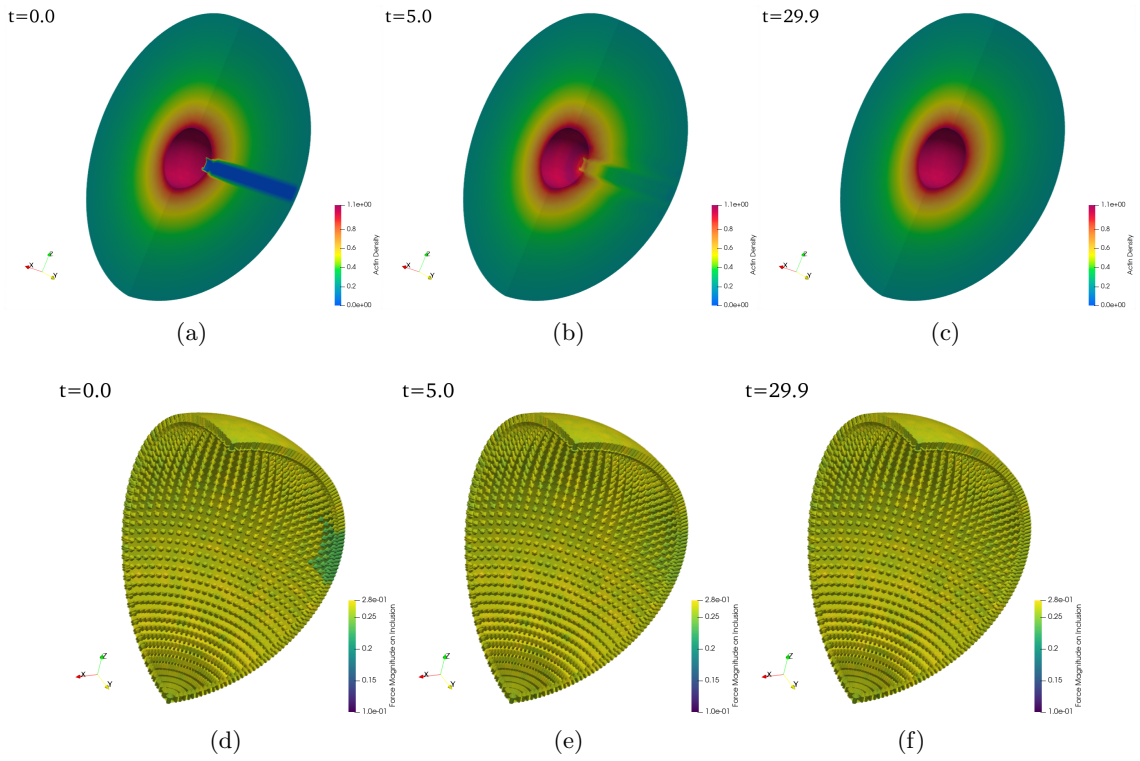



Figure 5.7: Temporal progression of the actin density distribution and force density on the inclusion after ablating actin concentration (a quarter-section of the spherical shell is depicted). **(a)** Initial ablated state with zero density in blue. Actin density profile at $t = 0$. **(b)** Recovered actin density at $t = 5.0$. **(c)** Recovered steady-state actin density at $t = 29.9$. **(d)** Force density on the inclusion (inner spherical shell). Inhomogeneous force distribution after ablation at $t = 0$. **(e)** Force density on the inclusion at $t = 5.0$. **(f)** Force density on the inclusion at $t = 29.9$.

**Fixed velocity boundary conditions.** In this case we impose:

$$\mathbf{v}(\mathbf{r} \in \mathcal{S}_{1,2}, t) = \mathbf{v}_{1,2}\,, \tag{5.9}$$

where $\mathbf{v}_{1,2}$ are the prescribed velocity fields at the inclusion and outer surfaces respectively. In the spherically symmetric case, these boundary conditions simplify to:

$$v(r = R_{1,2}, t) = v_{1,2}\,. \tag{5.10}$$

**Stress-free boundary conditions.** Alternatively, we can impose a vanishing stress at the inclusion and outer surfaces. We define $\mathbf{n}_{1,2}$ the surface normals at the inclusion and at the outer surface respectively, and we impose:

$$\sigma(\mathbf{r} \in \mathcal{S}_{1,2}, t) \cdot \mathbf{n}_{1,2} = 0\,. \tag{5.11}$$

In the spherically symmetric case, these boundary conditions read:

$$\sigma_{rr}(r = R_{1,2}, t) = 0\,, \tag{5.12}$$

with $\sigma_{rr} = 4\eta(\partial_r v - v/r)/3 + \bar{\eta}(\partial_r v + 2v/r) - P$.

### 5.4.3 Simulation results

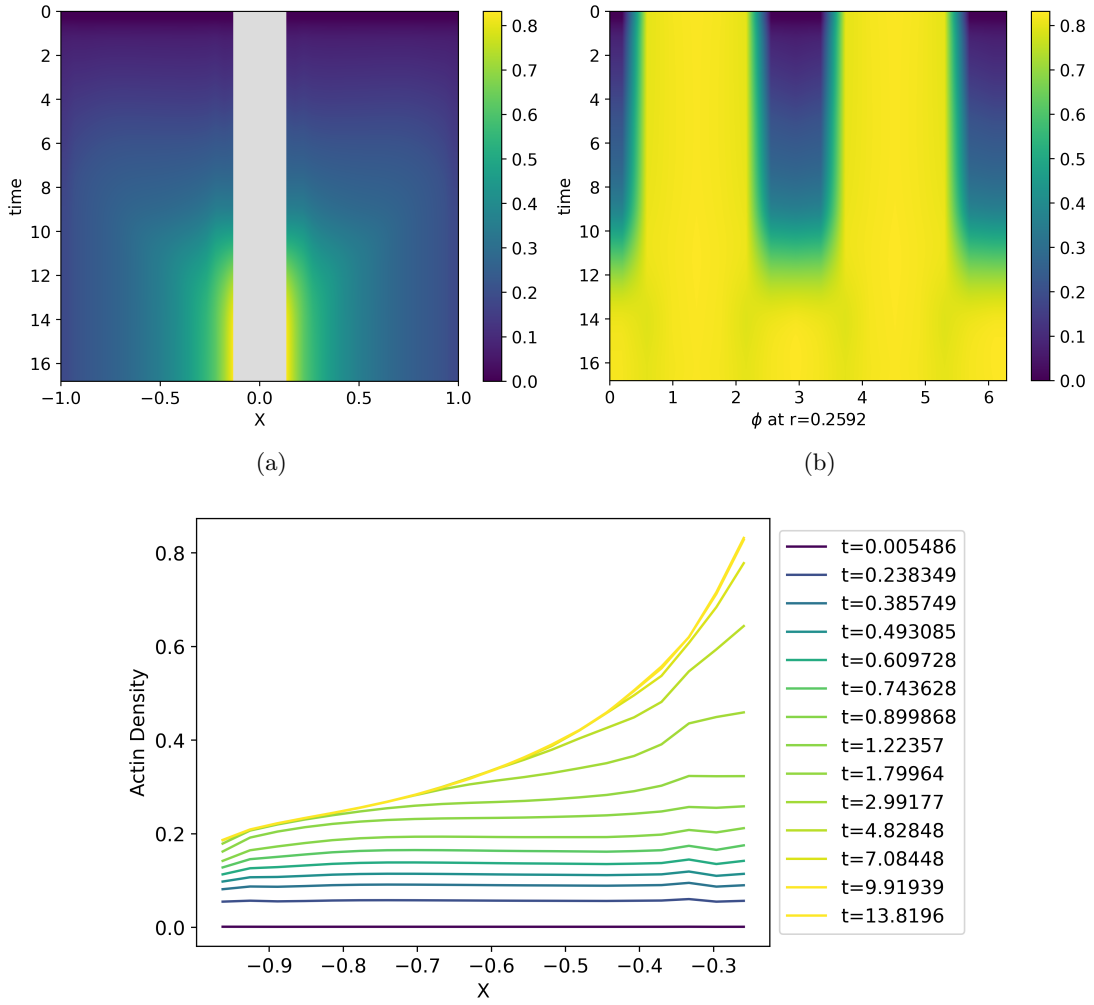
We perform the numerical simulations of the Eqs. (5.1, 5.2) in a full 3D spherical shell domain. More numerical details can be found in Appendix C.

#### Steady-state actin flow

We observe a steady-state flow and density distribution as shown in Fig. 5.6c with the actin density profile conforming to the experimental fit obtained from one-dimensional spherically symmetric model as shown in Fig. 5.5c. The convergence plot is shown in Appendix Fig. C.1. We further test our three-dimensional simulation by initializing it with a spatially inhomogeneous actin density profile that is not spherically symmetric. We find the same steady state as before in Fig. 5.6a-c, suggesting actin turnover and hydrodynamics as a robust mechanism for actin reorganization.

#### Laser ablation simulation

Next, akin to the experiments of laser ablation of the droplets, we simulate the dynamics with an ablated initial actin density and compute the force distribution on the inclusion. The results of the simulations are shown in Fig. 5.7. Experimentally, it was found that the inclusion compresses in the direction perpendicular to the axis of ablation. In the simulation, we find a decrease in the force density, suggesting imbalance of forces and a



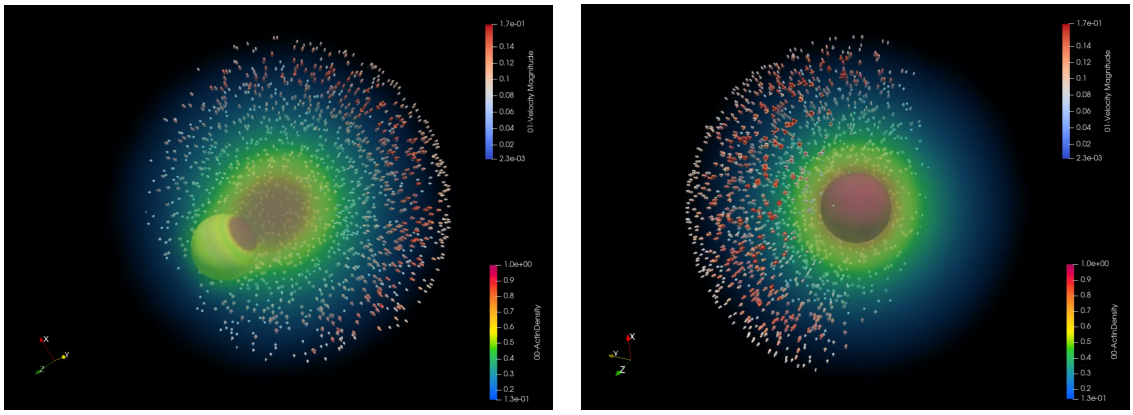

(a) (b)

(c)

Figure 5.8: Kymographs of the actin density recovery after ablation. **(a)** the kymograph in the radial direction along the axis of ablation. **(b)** Kymograph in the XY plane with $r = 0.26$. **(c)** Actin density profiles in time along the radial ablation axis in (a).

possibility of the compression if the inclusion is modeled as a soft material instead of a hard sphere.

We further examine the kymograph of the actin density recovery along the ablation axis and the rotational theta axis. This is shown it in Fig. 5.8. The simulations suggest that a critical threshold of actin density is uniformly polyermized first (see Fig. 5.7c) before the



(a) (b)

Figure 5.9: Moving the inclusion with the restoring force. **(a)** Numerical solution of the initial steady-state of the actin density with the inclusion moved to an off-center location. The arrows indicate the flow velocity field. **(b)** Final steady state of the actin density with inclusion restored to the center location. The arrows indicate the flow velocity field.

actin starts flowing towards the center due to the active hydrodynamics.

### Moving inclusion

Assume a magnetic bead is injected in to the droplet, the centering force would keep it inside the inclusion. Magnetic tweezers could then be used to perturb the system and move the entire inclusion to an off-center location. Under the assumption that the actin flow is relaxing much faster than the dynamics of the inclusion, we simulate the active hydrodynamics with the inclusion moving with a velocity computed from the net non-zero force on the inclusion when it is at an off-center location. The dynamics is simulated until steady-state is reached. We find that the inclusion is restored to the center location as also observed in the experiment and shown in Fig. 5.9a,b.

This suggests that the active hydrodynamic model can quantitatively capture the dynamics of such actin cytoskeleton extracts. Further, it shows the use of the present numerical techniques and high-performance computer codes for a scalable 3D simulation of active hydrodynamics in complex geometries.

## 5.5 Conclusion

This chapter showed the application of the present simulation framework to active hydrodynamics in complex geometries. We focused on the active nematic model as introduced in chapter 3 and numerically established the existence of the bend flow instability in spherical domains. This is in direct agreement with experimental observations

that indicate the existence of a critical activity or critical confining radius above which the system becomes unstable. We compared Dirichlet and Neumann boundary conditions for the polarity field where we found that both boundary conditions result in a similar spontaneous flow behavior above a critical radius or activity. The flow states consists of an initial spontaneous flow transition, followed by oscillatory and chaotic flows on increasing activity or the radii. We presented a numerical phase diagram of the instability, suggesting a Fréedericksz transition is also possible in a 3D ball, similar to the simulations in 3D channels in the previous chapter.

Next, we showcase our numerical solver by solving the active Ericksen-Leslie hydrodynamic equations in 3D annular domains and "peanut" shaped geometries. For the annulus, we found coherent flows as reported in the literature. In the future, our simulations could be used as a reference to design active engines. Further, the "peanut" case shows that our numerical solver is robust to geometric deformations. This property is extremely useful for a future solver with deforming domains to model morphogenesis.

Finally, we simulated cytoskeletal actin flows, specifically relating to the complex behavior observed in *Xenopus* egg cytoskeletal extracts. When these extracts are confined to water droplets immersed in oil, they display intriguing active hydrodynamic properties. An inclusion forms spontaneously at the center of the droplet, driven by the actomyosin spontaneous flows. The objective was to connect 3D active matter with these biological manifestations, emphasizing the hydrodynamics of cytoskeletal actin flows. In these experimental observations, no nematic ordering was observed. Therefore, to model this phenomenon, we considered an isotropic active gel model based on density. We examined a spherical water-based droplet of radius $R_2$, containing a contractile actomyosin network and surrounded by oil. Inside this droplet, we define the inclusion composed of displaced debris as a rigid sphere with radius $R_1 < R_2$. We modeled the contractile actomyosin network as a one-component active viscous fluid composed of the polymerized actin, characterized by mass density $\rho$ and velocity $\mathbf{v}$.

We simplified the discussion by considering the spherically symmetric case. The symmetric case is used to validate the full 3D model and fit the experimental data. Next, different boundary conditions necessary to solve the system of nonlinear differential equations were compared. These boundary conditions are required at the boundary with the outer droplet and at the boundary with inclusion. We discussed options for both fixed actin concentration and velocity, and for stress-free conditions at both the inclusion and outer surfaces. We selected the boundary conditions as per the experimental observations.

Finally, we numerically solved the model using the techniques developed in the previous chapters and showed the ability of the model to quantitatively predict the experimental dynamics. This further confirm the generality of the present framework for numerical simulations in non-Cartesian three-dimensional domains. This has the potential of advancing our understanding of active flows in biological systems, specifically of the hydrodynamics of cytoskeletal flows.

# Chapter 6

# Conclusion and future vision

## 6.1 Summary

We have presented a comprehensive exploration into the complex world of numerically solving the spatio-temporal Partial Differential Equations (PDEs) relevant to the study of active hydrodynamics. The studies presented in this thesis expanded our understanding of active hydrodynamics in complex geometries, through the development of new algorithms and models for numerical simulations. The main contribution was bridging the gap between mathematical models, physical principles, and biological phenomena through the development of novel supercomputing simulation algorithms.

The key contribution of the present work is an efficient and accessible framework for solving the hydrodynamics for various models in complex geometries using new meshless numerical methods and C++ template meta-programming with the scalable high-performance scientific computing library OpenFPM.

In the second chapter, we delved into the use of C++ expression templates, focusing on the design abstractions embedded in the OpenFPM library. Our work resulted in an expression system allowing for near-mathematical coding notation of the PDE, thereby fostering an open and scalable scientific computing environment. The practical relevance of this method, along with its efficiency and accessibility, is subsequently demonstrated by comparison with traditional coding approaches. We introduced a framework combining the advantages of a C++ expression system with a custom state type and distributed algebra system. This effectively extended the application of the Boost Odeint library for distributed-memory CPU and GPU high-performance computing (HPC) systems in OpenFPM. Our experiments with various biological hydrodynamics simulations showcased the potential of this framework to deliver high parallel efficiency. In all cases, we show the meshless method DC-PSE to be numerically convergent for solving biophysical hydrodynamics.

However, the present system is limited to currently available numerical methods in the OpenFPM numerics (sub)library and required slightly longer compilation times. Compile

time error messages can also be quite difficult to comprehend, due to the nested and lengthy nature of the expressions type trees. However, debugging the code is much easier because of the simpler mathematical notation used to write the codes. Further, the framework in the future could offer auto-tuning, for selection and tuning of the numerical methods [110].

In Chapter 3, we presented active polar fluids in three dimensions, introducing a generic hydrodynamic theory and proposing novel particle and hybrid particle-mesh algorithms for numerically solving the hydrodynamics of active matter. This chapter successfully applied the expression system, formulated in Chapter 2, to high-performance implementations, overcoming the *curse of dimensionality* associated to three-dimensional problems like lengthy equations, long simulation and computer code development time.

The key contribution here was a new algorithm to solve the three-dimensional active Ericksen-Leslie equations, a symmetry-preserving model for active polar fluids. This algorithm incorporates a hybrid particle-mesh method, demonstrating a notable advancement in the understanding of three-dimensional active fluid hydrodynamics. Using the meshless method Discretization-Corrected Particle Strength Exchange (DC-PSE) simplified the imposition of boundary conditions in complex domains and enabled the the arbitrary higher-order discretization of differential operators on unstructured particle distributions.

The use of the OpenFPM framework and the PDEs template expression system facilitated tackling computationally demanding 3D active matter problems. The algorithm's validity was verified in a series of benchmark simulations, which showed excellent parallel scaling and correctly converged to analytical solutions in active films with mixed Robin boundary conditions. This endorses the potential of our methodology to study active matter-related problems in three-dimensional domains.

However, the scalability of our algorithm is limited by the pressure correction scheme used to solve the incompressible force balance, with the linear system solver as the primary bottleneck. OpenFPM's modularity allows integrating alternative libraries for solving a linear system after matrix assembly. This implies that a future method with better scalability could potentially enhance our approach's parallel efficiency. The modular design also facilitates adaptability and versatility for changing requirements such as the use of a future direct solver for better numerical accuracy. Further, an alternative iterative solver from a different library could be used, if the current GMRES iterative solver fails to converge.

Another key contribution was the introduction of a novel mesh-free collocation method, Surface DC-PSE, for efficiently approximating intrinsic differential operators on curved surfaces. We benchmarked the method by discretizing the Laplace-Beltrami operator on a circular and spherical domain. Surface DC-PSE effectively leads to a surface stencil on surface points. We showed that this surface stencil can be used to solve implicit equations such as the Poisson equation. The surface stencil also results in a lower-dimensional sparse system, due to few stencil coefficients, as the only points required to evaluate the operator are the surface points in a small neighborhood. Despite the computational intensity of initially determining the stencil weights and the requirement of the normal field as input, Surface

DC-PSE allows for consistent higher-order numerical discretization of surface differential operators.

Overall, Chapter 3 presented new methods for numerically studying the hydrodynamics of active fluids in domains with curved boundaries and on surfaces. Our work allows for solving active hydrodynamics in 3D geometries, laying a solid foundation for future works using meshless numerical methods.

In Chapter 4, we presented numerical and analytical results for active hydrodynamics in 3D channels. The numerical simulation algorithm established in Chapter 3 was used to obtain the results. A significant discovery of this chapter was the spontaneous flow transition in 3D active fluids and the elucidation of its behavior under diverse boundary conditions and the sign of the active stress (contractile vs. extensile). With perpendicular anchoring of polarity at the boundary of a confining Y axes, this transition marks a significant departure from the 2D case, inducing a shear flow along both the X and Z axes. This complexity introduced by out-of-plane perturbations sets the stage for an in-depth exploration of active fluid dynamics in application areas such as cellular organism movement and novel materials development.

When parallel anchoring is imposed at the wall, the transition behaves differently. Then, contractile active stress incites counteracting out-of-plane perturbations, leading to an invariant extension of the 2D spontaneous flow transition. This emphasizes the role of boundary conditions in the dynamic behavior of active fluids.

An intriguing observation was the unique "wrinkling" phenomenon under extensile active stress, absent in 2D domains with rigid boundaries. This "wrinkling" adds another layer of complexity to the active fluid dynamics and can explain experimental observations of the instabilities of 3D microtubule assays with extensile active stress.

The simulation of active fluids of higher activity levels revealed a spectrum of intricate states, transitioning from traveling waves to intermittency, culminating in spatiotemporal chaos. These states resonate with patterns found in the well-known Berezinskii–Kosterlitz–Thouless (BKT) phase transitions. This correlation hints a richer connection between standard vector lattice models and active fluids with a BKT-like topological phase transition. This suggests active fluids first undergo a spontaneous flow transition, followed by a topological phase transition on increasing activity.

In conclusion, Chapter 4 revealed the intricate dynamics and transitions of active fluids under varied conditions and activity levels. These findings provided a valuable foundation for both theoretical and practical advancements in the study of active fluids.

Chapter 5 was devoted to the numerically studying the hydrodynamics of active fluids in complex 3D geometries, comparing different active fluid models. In this chapter, we numerically established the presence of a bend flow instability within a spherical domain. Our results are consistent with experimental observations that point to the existence of a critical activity or a critical confining radius, beyond which the system becomes unstable. We numerically found this critical threshold to be much higher when compared to the 3D channels. This provided a valuable understanding of how the hydrodynamic stability

of these systems can be controlled. The comprehensive phase diagram further suggested the existence of an active Fréedericksz transition in a 3D ball. Further, we showed the applicability of the meshless numerical solver in 3D annular domains. Those simulations showed that three-dimensional active matter can exhibit coherent circular motion in a three-dimensional annulus. We further showcased the simulations in a "peanut" geometry that resembles a dividing cell, where chiral flow vortices appear.

A key focus in this chapter was the study of cytoskeletal actin flows, particularly concerning the complex behaviors observed in *Xenopus* egg cytoskeletal extracts. When confined to water droplets submerged in oil, these extracts display fascinating active hydrodynamic properties, including the spontaneous formation of an opaque inclusion at the droplet's center, driven by spontaneous flows of actomyosin. In those experiments, no nematic order was observed in the active matter. Thus, we considered an isotropic active gel model using density to study the spontaneous flow mechanism.

We described a model within a spherical water-based droplet containing a contractile actomyosin network and surrounded by oil. Within this droplet, we defined the inclusion composed of displaced debris as a rigid sphere. The contractile actomyosin network was modeled as a one-component active viscous fluid composed of polymerized actin, characterized by mass density and velocity. A one-dimensional model was used to fit the experimental data, and as a benchmark for the following full 3D simulation with the inclusion in the center being exactly equivalent to the spherically symmetric case.

Finally, we numerically solved the model in 3D using the techniques developed in the previous chapters, demonstrating the model's ability to quantitatively predict experimental dynamics. This showed the applicability of the simulation framework developed for complex three-dimensional simulations. Overall, this chapter advanced our understanding of active matter in biological systems, with a particular emphasis on the hydrodynamics of cytoskeletal flows.

## 6.2 Vision and outlook

In the future, we intend to expand the expression system to incorporate more numerical methods and general meshes, making the framework more versatile. The user experience can be further improved by providing scripting language wrappers and integrating the system with domain-specific simulation languages [138, 139]. Ultimately, our goal is to evolve the framework into a robust tool for researchers, enabling them to solve complex PDEs efficiently and flexibly. The C++ template expression system, as a part of the OpenFPM library, is open for exploration and use, paving the way for significant advancements in the scientific exploration physical models. Our simulation techniques and the expression system could be used to solve discrete models such as Brownian rods with active diffusivity or kinetic theories. The implementation in OpenFPM will transparently allow for implementation of scalable simulation code of such models in an easier manner.

Looking ahead, we believe that the methods and insights provided in this thesis lay a strong foundation for future research in various directions. Our numerical algorithms, including the novel Surface DC-PSE method, can serve as robust tools for studying a multitude of partial differential equations not only in active matter, but also in other fields where such nonlinear dynamics play a crucial role in modeling the system. We believe that these methods can be further refined, extended, and integrated into existing computational tools to handle even more complex and diverse problems that involve deforming domains. We envision Surface DC-PSE serving as a useful tool to solve problems involving moving and deforming surfaces in the Lagrangian frame of reference. Future research will also include exploring the potential of coupling Surface DC-PSE with regular DC-PSE in surrounding space. We could compute covariant derivatives on hypersurfaces in three or more dimensions by using the Surface DC-PSE method. The scalable implementation will address the higher dimensionality through distributed computing.

Our study of active hydrodynamics in different geometries, from 3D channels, spherical balls, annulus to "peanut" like shapes, has uncovered a host of intriguing dynamics and transitions. The additional complexity introduced by out-of-plane perturbations, and the unique "wrinkling" phenomenon observed under extensile active stress, are examples of new behaviors that our work has brought to light such as the BKT-like topological phase transition. We envision these insights inspiring more in-depth studies on the fundamental principles governing active fluid dynamics and their manifestations under different conditions. In future, we expect to create a detailed understanding of the chaotic state with accurate statistical predictions about the dynamics of defect loops. In the chaotic state, defect loops appear spontaneously, and are topologically active. They display complex braiding dynamics in our simulations. Our simulations could used to build a comprehensive understanding of 3D active nematic suspensions. Furthermore, our models and numerical simulations can be used to understand various macroscopic parameters, e.g., $\lambda$ that couples polarity to the active potential, and $\zeta$ that couples the active stress to the active potential. We have shown that these parameters play a governing role in the dynamics of active microtubule mixtures. Understanding their physical meaning i.e., by experimentally measured values, we will build a comprehensive understanding of controlling such inherently unstable systems and possibly uncover new physical mechanism that govern the macroscopic behavior of such systems.

The isotropic active gel model we developed for studying cytoskeletal actin flows provides a valuable stepping stone for building a more comprehensive understanding of active systems within biological organisms. Given the growing interest in the field of cytoskeletal dynamics in various biological processes, from cell migration to tissue development, our work has the potential to stimulate further experiments leading to a detailed model of morphogenesis. Future work could further elucidate these complex processes and extend them to other active systems, potentially leading to novel applications in the realm of developmental biology.

The work presented in this thesis also opens the doors for interdisciplinary studies, where insights from the field of active matter could inform developments in material science,

soft robotics, and even the design of micro and nano-scale machines. For instance, the observed "wrinkling" phenomenon could be harnessed in the design of smart materials that exhibit programmed deformations in response to external stimuli. The chaotic state could be used for mixing materials at the nano-scale.

In conclusion, while our work represents a significant step forward in understanding active hydrodynamics, we believe it is just the beginning. The algorithmic framework, universal methods, and application results we have established provide a launchpad for many exciting avenues of future research, spanning the realms of computer science, physics, mathematics, and biology. As we continue to refine our models and simulation techniques, we look forward to discovering even more fascinating dynamics and behaviors of active fluids, and to using these insights to innovate in a wide range of applications.

# Bibliography


[1] I. F. Sbalzarini, "Modeling and simulation of biological systems from image data," *BioEssays*, vol. 35, no. 5, pp. 482–490, 2013. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bies.201200051.

[2] H. Kitano, "Systems biology: A brief overview," *Science*, vol. 295, no. 5560, pp. 1662–1664, 2002.

[3] H. Kitano, "Computational systems biology," *Nature*, vol. 420, pp. 206–210, Nov. 2002.

[4] G. J. Stephens, B. Johnson-Kerner, W. Bialek, and W. S. Ryu, "Dimensionality and Dynamics in the Behavior of C. elegans," *PLoS Comput Biol*, vol. 4, p. e1000028, Apr. 2008.

[5] M. Buchanan, "The joy of simulation," *Nature Physics*, vol. 2, pp. 495–495, Aug. 2006.

[6] A. Mietke, V. Jemseena, K. V. Kumar, I. F. Sbalzarini, and F. Jülicher, "Minimal Model of Cellular Symmetry Breaking," *Phys. Rev. Lett.*, vol. 123, p. 188101, Oct. 2019. Publisher: American Physical Society.

[7] F. J. Boge, "Why computer simulations are not inferences, and in what sense they are experiments," *European Journal for Philosophy of Science*, vol. 9, Nov. 2018.

[8] R. Alert and X. Trepat, "Physical models of collective cell migration," *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 77–101, 2020.

[9] P. Chandrakar, M. Varghese, S. Aghvami, A. Baskaran, Z. Dogic, and G. Duclos, "Confinement Controls the Bend Instability of Three-Dimensional Active Liquid Crystals," *Phys. Rev. Lett.*, vol. 125, p. 257801, Dec. 2020. Publisher: American Physical Society.

[10] M. Varghese, A. Baskaran, M. F. Hagan, and A. Baskaran, "Confinement-induced self-pumping in 3d active fluids," *Physical Review Letters*, vol. 125, 12 2020.

[11] R. Farhadifar, J.-C. Röper, B. Aigouy, S. Eaton, and F. Jülicher, "The influence of cell mechanics, cell-cell interactions, and proliferation on epithelial packing," *Current Biology*, vol. 17, pp. 2095–2104, Dec. 2007.

[12] M. Popović, V. Druelle, N. A. Dye, F. Jülicher, and M. Wyart, "Inferring the flow properties of epithelial tissues from their geometry," *New Journal of Physics*, vol. 23, p. 033004, Mar. 2021.

[13] T. H. Tan, A. Amiri, I. Seijo-Barandiarán, M. F. Staddon, A. Materne, S. Tomas, C. Duclut, M. Popović, A. Grapin-Botton, and F. Jülicher, "Emergent chirality in active solid rotation of pancreas spheres," Sept. 2022.

[14] H. Margenau, G. M. Murphy, *et al.*, "Mathematics of physics and chemistry," 1956.

[15] G. B. Folland, *Introduction to partial differential equations*, vol. 102. Princeton university press, 1995.

[16] U. Ghia, K. N. Ghia, and C. T. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *Journal of Computational Physics*, vol. 48, pp. 387–411, Dec. 1982.

[17] R. Ramaswamy, G. Bourantas, F. Jülicher, and I. F. Sbalzarini, "A hybrid particle-mesh method for incompressible active polar viscous gels," *Journal of Computational Physics*, vol. 291, pp. 334–361, June 2015.

[18] S. Götschel and M. Weiser, "Compression challenges in large scale partial differential equation solvers," *Algorithms*, vol. 12, p. 197, Sep 2019.

[19] G. Duclos, C. Blanch-Mercader, V. Yashunsky, G. Salbreux, J.-F. Joanny, J. Prost, and P. Silberzan, "Spontaneous shear flow in confined cellular nematics," *Nature Physics*, vol. 14, pp. 728–732, Apr. 2018.

[20] G. Sarfati, A. Maitra, R. Voituriez, J.-C. Galas, and A. Estevez-Torres, "Crosslinking and depletion determine spatial instabilities in cytoskeletal active matter," *Soft Matter*, vol. 18, pp. 3793–3800, May 2022. Publisher: The Royal Society of Chemistry.

[21] M. C. Marchetti, J. F. Joanny, S. Ramaswamy, T. B. Liverpool, J. Prost, M. Rao, and R. A. Simha, "Hydrodynamics of soft active matter," *Rev. Mod. Phys.*, vol. 85, pp. 1143–1189, July 2013. Publisher: American Physical Society.

[22] F. Jülicher, S. W. Grill, and G. Salbreux, "Hydrodynamic theory of active matter," *Rep. Prog. Phys.*, vol. 81, p. 076601, June 2018.

[23] S. Ramaswamy, "Active matter," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2017, p. 054002, May 2017.

[24] T. G. C. Sydney Chapman, *The Mathematical Theory of Non-uniform Gases: An Account of the Kinetic Theory of Viscosity, Thermal Conduction and Diffusion in Gases*. Cambridge, England: Cambridge University Press, Apr. 1990.

[25] M. C. Marchetti, Y. Fily, S. Henkes, A. Patch, and D. Yllanes, "Minimal model of active colloids highlights the role of mechanical interactions in controlling the emergent behavior of active matter," *Current Opinion in Colloid and Interface Science*, vol. 21, pp. 34–43, 2016.

[26] L. Onsager, "Reciprocal relations in irreversible processes. ii.," *Physical Review*, vol. 38, pp. 2265–2279, 12 1931.

[27] N. W. Goehring, P. K. Trong, J. S. Bois, D. Chowdhury, E. M. Nicola, A. A. Hyman, and S. W. Grill, "Polarization of PAR Proteins by Advective Triggering of a Pattern-Forming System," *Science*, vol. 334, pp. 1137–1141, Nov. 2011. Publisher: American Association for the Advancement of Science Section: Report.

[28] L. Wettmann and K. Kruse, "The min-protein oscillations in escherichia coli: an example of self-organized cellular protein waves," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 373, p. 20170111, Apr. 2018.

[29] B. Ramm, T. Heermann, and P. Schwille, "The e. coli MinCDE system in the regulation of protein patterns and gradients," *Cellular and Molecular Life Sciences*, vol. 76, pp. 4245–4273, July 2019.

[30] J. Denk, S. Kretschmer, J. Halatek, C. Hartl, P. Schwille, and E. Frey, "Mine conformational switching confers robustness on self-organized min protein patterns," *Proceedings of the National Academy of Sciences*, vol. 115, no. 18, pp. 4553–4558, 2018.

[31] F. Brauns, G. Pawlik, J. Halatek, J. Kerssemakers, E. Frey, and C. Dekker, "Bulk-surface coupling identifies the mechanistic connection between min-protein patterns in vivo and in vitro," *Nature Communications*, vol. 12, June 2021.

[32] T. Sanchez, D. T. Chen, S. J. Decamp, M. Heymann, and Z. Dogic, "Spontaneous motion in hierarchically assembled active matter," *Nature*, vol. 491, pp. 431–434, 11 2012.

[33] A. Bacanu, J. F. Pelletier, Y. Jung, and N. Fakhri, "Inferring scale-dependent non-equilibrium activity using carbon nanotubes," *Nature Nanotechnology*, May 2023.

[34] D. Dell'Arciprete, M. L. Blow, A. T. Brown, F. D. C. Farrell, J. S. Lintuvuori, A. F. McVey, D. Marenduzzo, and W. C. K. Poon, "A growing bacterial colony in two dimensions as an active nematic," *Nature Communications*, vol. 9, Oct. 2018.

[35] T. H. Tan, J. Liu, P. W. Miller, M. Tekant, J. Dunkel, and N. Fakhri, "Topological turbulence in the membrane of a living cell," *Nature Physics*, vol. 16, pp. 657–662, Mar. 2020.

[36] G. Popkin, "The physics of life," *Nature*, vol. 529, pp. 16–18, Jan. 2016.

[37] S. F. Gilbert, *Developmental Biology*. Sunderland, MA: Sinauer Associates, 11 ed., June 2016.

[38] A. Mongera, P. Rowghanian, H. J. Gustafson, E. Shelton, D. A. Kealhofer, E. K. Carn, F. Serwane, A. A. Lucio, J. Giammona, and O. Campàs, "A fluid-to-solid jamming transition underlies vertebrate body axis elongation," *Nature*, vol. 561, pp. 401–405, Sept. 2018.

[39] M. Belz, L. W. Pyritz, and M. Boos, "Spontaneous flocking in human groups," *Behavioural Processes*, vol. 92, pp. 6–14, 2013.

[40] E. Schrödinger, *What Is Life? The Physical Aspect of the Living Cell*. Cambridge, England: Cambridge University Press, 1944.

[41] N. Fakhri, A. D. Wessel, C. Willms, M. Pasquali, D. R. Klopfenstein, F. C. MacKintosh, and C. F. Schmidt, "High-resolution mapping of intracellular fluctuations using carbon nanotubes," *Science*, vol. 344, pp. 1031–1035, May 2014.

[42] M. R. Shaebani, A. Wysocki, R. G. Winkler, G. Gompper, and H. Rieger, "Computational models for active matter," *arXiv:1910.02528 [cond-mat, physics:physics]*, Oct. 2019. arXiv: 1910.02528.

[43] B. Najma, M. Varghese, L. Tsidilkovski, L. Lemma, A. Baskaran, and G. Duclos, "Competing instabilities reveal how to rationally design and control active crosslinked gels," *Nat Commun*, vol. 13, p. 6465, Oct. 2022. Number: 1 Publisher: Nature Publishing Group.

[44] N. Hu, F. Ma, and Y. Wang, "Motion of a self-propelled rod with brownian and hydrodynamics interactions," *Journal of Physics: Conference Series*, vol. 2012, p. 012001, Sept. 2021.

[45] A. R. Sprenger, M. A. Fernandez-Rodriguez, L. Alvarez, L. Isa, R. Wittkowski, and H. Löwen, "Active brownian motion with orientation-dependent motility: Theory and experiments," *Langmuir*, vol. 36, pp. 7066–7073, Jan. 2020.

[46] A. Callegari and G. Volpe, "Numerical simulations of active brownian particles," in *Soft and Biological Matter*, pp. 211–238, Springer International Publishing, 2019.

[47] Y. Peng, L. Lai, Y.-S. Tai, K. Zhang, X. Xu, and X. Cheng, "Diffusion of ellipsoids in bacterial suspensions," *Phys. Rev. Lett.*, vol. 116, p. 068303, Feb 2016.

[48] T. Gao, M. D. Betterton, A.-S. Jhang, and M. J. Shelley, "Analytical structure, dynamics, and coarse graining of a kinetic model of an active fluid," *Phys. Rev. Fluids*, vol. 2, p. 093302, Sep 2017.

[49] D. B. Stein, G. De Canio, E. Lauga, M. J. Shelley, and R. E. Goldstein, "Swirling instability of the microtubule cytoskeleton," *Phys. Rev. Lett.*, vol. 126, p. 028103, Jan 2021.

[50] S. Weady, D. B. Stein, and M. J. Shelley, "Thermodynamically consistent coarse-graining of polar active fluids," *Phys. Rev. Fluids*, vol. 7, p. 063301, Jun 2022.

[51] S. Weady, M. J. Shelley, and D. B. Stein, "A fast chebyshev method for the bingham closure with application to active nematic suspensions," *Journal of Computational Physics*, vol. 457, p. 110937, 2022.

[52] S. Fürthauer, B. Lemma, P. J. Foster, S. C. Ems-McClung, C.-H. Yu, C. E. Walczak, Z. Dogic, D. J. Needleman, and M. J. Shelley, "Self-straining of actively crosslinked microtubule networks," *Nat. Phys.*, vol. 15, pp. 1295–1300, Dec. 2019. Number: 12 Publisher: Nature Publishing Group.

[53] F. Schwietert and J. Kierfeld, "Bistability and oscillations in cooperative microtubule and kinetochore dynamics in the mitotic spindle," *New Journal of Physics*, vol. 22, p. 053008, May 2020.

[54] W. Yan, S. Ansari, A. Lamson, M. A. Glaser, R. Blackwell, M. D. Betterton, and M. Shelley, "Toward the cellular-scale simulation of motor-driven cytoskeletal assemblies," *eLife*, vol. 11, p. e74160, may 2022.

[55] T. Strübing, A. Khosravanizadeh, A. Vilfan, E. Bodenschatz, R. Golestanian, and I. Guido, "Wrinkling Instability in 3D Active Nematics," *Nano Lett.*, vol. 20, pp. 6281–6288, Sept. 2020. Publisher: American Chemical Society.

[56] R. Zhang, Y. Zhou, M. Rahimi, and J. J. de Pablo, "Dynamic structure of active nematic shells," *Nature Communications*, vol. 7, Nov. 2016.

[57] S. Alam, B. Najma, A. Singh, J. Laprade, G. Gajeshwar, H. Yevick, A. Baskaran, P. Foster, and G. Duclos, "Active fréedericksz transition in three-dimensional active nematic droplets," Under Review.

[58] M. Malik-Garbi, N. Ierushalmi, S. Jansen, E. Abu-Shah, B. L. Goode, A. Mogilner, and K. Keren, "Scaling behaviour in steady-state contracting actomyosin networks," *Nat. Phys.*, vol. 15, no. 5, pp. 509–516, 2019.

[59] K. Kruse, J.-F. Joanny, F. Julicher, J. Prost, and K. Sekimoto, "Generic theory of active polar gels: a paradigm for cytoskeletal dynamics," *Eur. Phys. J. E*, vol. 16, pp. 5–16, Jan. 2005. arXiv: physics/0406058.

[60] K. Kruse, J. F. Joanny, F. Jülicher, J. Prost, and K. Sekimoto, "Asters, Vortices, and Rotating Spirals in Active Gels of Polar Filaments," *Phys. Rev. Lett.*, vol. 92, p. 078101, Feb. 2004.

[61] R. Voituriez, J. F. Joanny, and J. Prost, "Spontaneous flow transition in active polar gels," *Europhys. Lett.*, vol. 70, pp. 404–410, May 2005.

[62] A. Mietke, F. Jülicher, and I. F. Sbalzarini, "Self-organized shape dynamics of active surfaces," *PNAS*, vol. 116, pp. 29–34, Jan. 2019.

[63] A. J. Tan, E. Roberts, S. A. Smith, U. A. Olvera, J. Arteaga, S. Fortini, K. A. Mitchell, and L. S. Hirst, "Topological chaos in active nematics," *Nat. Phys.*, vol. 15, pp. 1033–1039, Oct. 2019. Number: 10 Publisher: Nature Publishing Group.

[64] A. Singh, P. Incardona, and I. F. Sbalzarini, "A C++ expression system for partial differential equations enables generic simulations of biological hydrodynamics," *Eur. Phys. J. E*, vol. 44, p. 117, Sept. 2021.

[65] M. J. Bowick, N. Fakhri, M. C. Marchetti, and S. Ramaswamy, "Symmetry, Thermodynamics and Topology in Active Matter," *arXiv:2107.00724 [cond-mat]*, July 2021. arXiv: 2107.00724.

[66] S. Maddu, D. Sturm, B. L. Cheeseman, C. L. Müller, and I. F. Sbalzarini, "Stencil-net: Data-driven solution-adaptive discretization of partial differential equations," 2021.

[67] S. A. Faroughi, N. Pawar, C. Fernandes, M. Raissi, S. Das, N. K. Kalantari, and S. K. Mahjour, "Physics-guided, physics-informed, and physics-encoded neural networks in scientific computing," 2022.

[68] A. Voigt, "Fluid deformable surfaces," *J. Fluid Mech.*, vol. 878, pp. 1–4, Nov. 2019.

[69] M. Nestler, I. Nitschke, and A. Voigt, "A finite element approach for vector- and tensor-valued surface PDEs," *Journal of Computational Physics*, vol. 389, pp. 48–61, July 2019.

[70] J. Pahlke and I. F. Sbalzarini, "A unifying mathematical definition of particle methods," *IEEE Open Journal of the Computer Society*, vol. 4, pp. 97–108, 2023.

[71] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars," *Monthly Notices of the Royal Astronomical Society*, vol. 181, pp. 375–389, Dec. 1977.

[72] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis," *The Astronomical Journal*, vol. 82, p. 1013, Dec. 1977.

[73] A. Monteleone, M. Monteforte, and E. Napoli, "Inflow/outflow pressure boundary conditions for smoothed particle hydrodynamics simulations of incompressible flows," *Computers & Fluids*, vol. 159, pp. 9–22, Dec. 2017.

[74] Q. W. Ma, Y. Zhou, and S. Yan, "A review on approaches to solving Poisson's equation in projection-based meshless methods for modelling strongly nonlinear water waves," *J. Ocean Eng. Mar. Energy*, vol. 2, pp. 279–299, Aug. 2016.

[75] J.-P. Fürstenau, B. Avci, and P. Wriggers, "A comparative numerical study of pressure-Poisson-equation discretization strategies for SPH," June 2017.

[76] J. P. Morris, P. J. Fox, and Y. Zhu, "Modeling Low Reynolds Number Incompressible Flows Using SPH," *Journal of Computational Physics*, vol. 136, pp. 214–226, Sept. 1997.

[77] B. Schrader, S. Reboux, and I. F. Sbalzarini, "Discretization correction of general integral PSE Operators for particle methods," *Journal of Computational Physics*, vol. 229, pp. 4159–4182, June 2010.

[78] G. C. Bourantas, B. L. Cheeseman, R. Ramaswamy, and I. F. Sbalzarini, "Using DC PSE operator discretization in Eulerian meshless collocation methods improves their robustness in complex geometries," *Computers & Fluids*, vol. 136, pp. 285–300, Sept. 2016.

[79] B. Schrader, S. Reboux, and I. F. Sbalzarini, "Choosing the Best Kernel: Performance Models for Diffusion Operators in Particle Methods," *SIAM J. Sci. Comput.*, vol. 34, pp. A1607–A1634, Jan. 2012.

[80] M. E. Cates, O. Henrich, D. Marenduzzo, and K. Stratford, "Lattice Boltzmann simulations of liquid crystalline fluids: active gels and blue phases," *Soft Matter*, vol. 5, pp. 3791–3800, Oct. 2009. Publisher: The Royal Society of Chemistry.

[81] M. Cates, O. Henrich, D. Marenduzzo, and K. Stratford, "Lattice Boltzmann simulations of liquid crystalline fluids: Active gels and blue phases," *Soft Matter*, vol. 5, Sept. 2010.

[82] L. N. Carenza, G. Gonnella, A. Lamura, G. Negro, and A. Tiribocchi, "Lattice boltzmann methods and active fluids," *The European Physical Journal E*, vol. 42, p. 81, 6 2019.

[83] C. Denniston, E. Orlandini, and J. M. Yeomans, "Simulations of liquid crystal hydrodynamics in the isotropic and nematic phases," *Europhysics Letters (EPL)*, vol. 52, pp. 481–487, 11 2000.

[84] C. Denniston, E. Orlandini, and J. M. Yeomans, "Lattice boltzmann simulations of liquid crystal hydrodynamics," *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, vol. 63, 2001.

[85] C. Denniston, D. Marenduzzo, E. Orlandini, and J. M. Yeomans, "Lattice boltzmann algorithm for three–dimensional liquid–crystal hydrodynamics," *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 362, pp. 1745–1754, 8 2004.

[86] D. Marenduzzo, E. Orlandini, M. E. Cates, and J. M. Yeomans, "Steady-state hydrodynamic instabilities of active liquid crystals: Hybrid lattice boltzmann simulations," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 76, 9 2007.

[87] D. Marenduzzo, E. Orlandini, and J. M. Yeomans, "Hydrodynamics and rheology of active liquid crystals: A numerical investigation," *Physical Review Letters*, vol. 98, 3 2007.

[88] L. Giomi, L. Mahadevan, B. Chakraborty, and M. F. Hagan, "Excitable patterns in active nematics," *Physical Review Letters*, vol. 106, 5 2011.

[89] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-gas automata for the navier-stokes equation," *Physical Review Letters*, vol. 56, 1986.

[90] C.-L. L. Yong G. Lai, Jianchun Huang, "Accuracy and Efficiency Study of Lattice Boltzmann Method for Steady-State Flow Simulations," *Numerical Heat Transfer, Part B: Fundamentals*, vol. 39, pp. 21–43, Jan. 2001. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/104077901460669.

[91] J. C. Butcher, "A history of runge-kutta methods," *Applied numerical mathematics*, vol. 20, no. 3, pp. 247–260, 1996.

[92] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, "Implicit-explicit runge-kutta methods for time-dependent partial differential equations," *Applied Numerical Mathematics*, vol. 25, pp. 151–167, Nov. 1997.

[93] W. E. Nagel, D. H. Kroener, and M. M. Resch, eds., *High performance computing in science and engineering ' 18*. Cham, Switzerland: Springer Nature, 1 ed., June 2019.

[94] "Mpi: A message passing interface," in *Supercomputing '93:Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pp. 878–883, 1993.

[95] H. Jasak, "OpenFOAM: Open source CFD in research and industry," *International Journal of Naval Architecture and Ocean Engineering*, vol. 1, pp. 89–94, Dec. 2009.

[96] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, "The fenics project version 1.5," *¡p¿Archive of Numerical Software*, vol. Vol 3, p. ¡strong¿Starting Point and Frequency: ¡/strong¿Year: 2013¡/p¿, 2015.

[97] K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet, and B. P. Brown, "Dedalus: A flexible framework for numerical simulations with spectral methods," *Physical Review Research*, vol. 2, Apr. 2020.

[98] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries," in *Modern Software Tools for Scientific Computing* (E. Arge, A. M. Bruaset, and H. P. Langtangen, eds.), pp. 163–202, Boston, MA: Birkhäuser, 1997.

[99] T. Trilinos Project Team, *The Trilinos Project Website*, 2020 (acccessed May 22, 2020).

[100] T. A. Davis, "Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method," *ACM Trans. Math. Softw.*, vol. 30, p. 196–199, jun 2004.

[101] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[102] P. Incardona, A. Leo, Y. Zaluzhnyi, R. Ramaswamy, and I. F. Sbalzarini, "OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers," *Computer Physics Communications*, vol. 241, pp. 155–177, Aug. 2019.

[103] P. Incardona, T. Bianucci, and I. F. Sbalzarini, "Distributed sparse block grids on GPUs," in *Lecture Notes in Computer Science*, pp. 272–290, Springer International Publishing, 2021.

[104] P. Incardona, A. Gupta, S. Yaskovets, and I. F. Sbalzarini, "A c++ library for memory layout and performance portability of scientific applications," in *Euro-Par 2022: Parallel Processing Workshops*, pp. 109–120, Springer Nature Switzerland, 2023.

[105] P. Incardona, A. Gupta, S. Yaskovets, and I. F. Sbalzarini, "A portable c++ library for memory and compute abstraction on multi-core cpus and gpus," *Concurrency and Computation: Practice and Experience*, vol. n/a, no. n/a, p. e7870.

[106] C. Preundl, B. Bergen, F. Hülsemann, and U. Rüde, "ParEXPDE: Expression templates and advanced PDE software design on the Hitachi SR8000," in *High Performance Computing in Science and Engineering, Garching 2004*, pp. 167–179, Springer, 2005.

[107] G.-H. Cottet, "A particle-grid superposition method for the Navier-Stokes equations," *Journal of Computational Physics*, vol. 89, pp. 301–318, Aug. 1990.

[108] T. Veldhuizen, "Expression templates," *C++ Report*, vol. 7, pp. 26–31, 1995.

[109] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[110] N. Khouzami, F. Michel, P. Incardona, J. Castrillon, and I. F. Sbalzarini, "Model-based autotuning of discretization methods in numerical simulations of partial differential equations," *Journal of Computational Science*, vol. 57, p. 101489, 2022.

[111] J. C. Butcher, "A history of runge-kutta methods," *Applied numerical mathematics*, vol. 20, no. 3, pp. 247–260, 1996.

[112] W. H. Press and S. A. Teukolsky, "Adaptive stepsize runge-kutta integration," *Computers in Physics*, vol. 6, no. 2, pp. 188–191, 1992.

[113] K. Ahnert, M. Mulansky, T. E. Simos, G. Psihoyios, C. Tsitouras, and Z. Anastassi, "Odeint – solving ordinary differential equations in c++," in *AIP Conference Proceedings*, AIP, 2011.

[114] M. Ahnert, "ODEINT State types, algebras and operations." `https://www.boost.org/doc/libs/1_66_0/libs/numeric/odeint/doc/html/index.html`, 2015. [Online; accessed 20-April-2021].

[115] P. Incardona, A. Leo, Y. Zaluzhnyi, R. Ramaswamy, and I. F. Sbalzarini, "Openfpm: A scalable open framework for particle and particle-mesh codes on parallel computers," *Computer Physics Communications*, vol. 241, pp. 155–177, 2019.

[116] I. F. Sbalzarini, J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos, "PPM – a highly efficient parallel particle-mesh library for the simulation of continuum systems," *J. Comput. Phys.*, vol. 215, no. 2, pp. 566–588, 2006.

[117] D. I. Ketcheson, "Highly efficient strong stability-preserving runge–kutta methods with low-storage implementations," *SIAM Journal on Scientific Computing*, vol. 30, no. 4, pp. 2113–2136, 2008.

[118] R. T. Mills, M. F. Adams, S. Balay, J. Brown, A. Dener, M. Knepley, S. E. Kruger, H. Morgan, T. Munson, K. Rupp, B. F. Smith, S. Zampini, H. Zhang, and J. Zhang, "Toward performance-portable PETSc for GPU-based exascale systems," *Parallel Computing*, vol. 108, p. 102831, 2021.

[119] D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos, "Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers," *ACM Transactions on Mathematical Software (TOMS)*, 2022.

[120] A. C. Hindmarsh and L. R. Petzold, "LSODA, Ordinary Differential Equation Solver for Stiff or Non-Stiff System," Sept. 2005.

[121] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[122] A. Mohammadi, S. Mohamed, S. Nahavandi, and K. Ahnert, "CISR-ODE, A C++ Framework with ODE Solver for Code Based System Dynamics Simulation," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, (Kowloon Tong, Hong Kong), pp. 401–406, IEEE Press, Oct. 2015.

[123] P. Incardona, A. Gupta, S. Yaskovets, and I. F. Sbalzarini, "A c++ library for memory layout and performance portability of scientific applications," in *Euro-Par 2022: Parallel Processing Workshops*, (Cham), Springer International Publishing, 2023.

[124] P. Incardona, T. Bianucci, and I. F. Sbalzarini, "Distributed sparse block grids on GPUs," in *Proc. International Conference on High Performance Computing (ISC)*, vol. 12728 of *Lecture Notes in Computer Science*, (Cham, Switzerland), pp. 272–290, Springer, 2021.

[125] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[126] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[127] B. Schrader, S. Reboux, and I. F. Sbalzarini, "Discretization correction of general integral pse operators for particle methods," *Journal of Computational Physics*, vol. 229, no. 11, pp. 4159–4182, 2010.

[128] "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 237, pp. 37–72, Aug. 1952.

[129] A. M. Turing, "The chemical basis of morphogenesis," *Bltn Mathcal Biology*, vol. 52, pp. 153–197, Jan. 1990.

[130] J. Howard, S. W. Grill, and J. S. Bois, "Turing's next steps: the mechanochemical basis of morphogenesis," *Nat Rev Mol Cell Biol*, vol. 12, pp. 392–398, June 2011.

[131] Y. Liu, R. M. C. So, and C. H. Zhang, "Modeling the bifurcating flow in a human lung airway," *Journal of Biomechanics*, vol. 35, pp. 465–473, Apr. 2002.

[132] K. Perktold and G. Rappitsch, "Computer simulation of local blood flow and vessel mechanics in a compliant carotid artery bifurcation model," *Journal of Biomechanics*, vol. 28, pp. 845–856, July 1995.

[133] I. Vignon-Clementel, C. Figueroa, K. Jansen, and C. Taylor, "Outflow boundary conditions for three-dimensional finite element modeling of blood flow and pressure in arteries," *Computer Methods in Applied Mechanics and Engineering*, vol. 195, pp. 3776–3796, June 2006.

[134] H. Turlier, B. Audoly, J. Prost, and J.-F. Joanny, "Furrow constriction in animal cell cytokinesis," *Biophysical Journal*, vol. 106, no. 1, pp. 114–123, 2014.

[135] P. K. Papadopoulos, "An auxiliary potential velocity method for incompressible viscous flow," *Computers & Fluids*, vol. 51, pp. 60–67, Dec. 2011.

[136] G. C. Bourantas, V. C. Loukopoulos, E. D. Skouras, V. N. Burganos, and G. C. Nikiforidis, "An IPOT meshless method using DC PSE approximation for fluid flow equations in 2D and 3D geometries," *AIP Conference Proceedings*, vol. 1738, p. 480066, June 2016.

[137] G. C. Bourantas and V. C. Loukopoulos, "A meshless scheme for incompressible fluid flow using a velocity–pressure correction method," *Computers & Fluids*, vol. 88, pp. 189–199, Dec. 2013.

[138] N. Khouzami, L. Schütze, P. Incardona, L. Kraatz, T. Subic, J. Castrillon, and I. F. Sbalzarini, "The OpenPME Problem Solving Environment for Numerical Simulations," in *Computational Science – ICCS 2021* (M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, eds.), Lecture Notes in Computer Science, (Cham), pp. 614–627, Springer International Publishing, 2021.

[139] S. Karol, T. Nett, J. Castrillon, and I. F. Sbalzarini, "A Domain-Specific Language and Editor for Parallel Particle Methods," *ACM Trans. Math. Softw.*, vol. 44, no. 3, pp. 34:1–34:32, 2018.

[140] N. Khouzami, L. Schütze, P. Incardona, L. Kraatz, T. Subic, J. Castrillon, and I. F. Sbalzarini, "The OpenPME Problem Solving Environment for Numerical Simulations," in *Computational Science – ICCS 2021* (M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, eds.), Lecture Notes in Computer Science, (Cham), pp. 614–627, Springer International Publishing, 2021.

[141] B. Schrader, S. Reboux, and I. F. Sbalzarini, "Discretization correction of general integral pse operators for particle methods," *Journal of Computational Physics*, vol. 229, pp. 4159–4182, 6 2010.

[142] P. Incardona, A. Leo, and Y. Zaluzhnyi, "Openfpm: A scalable open framework for particle and particle-mesh codes on parallel computers," *Computer Physics Communications*, vol. 241, pp. 155–177, 2019.

[143] A. Singh, P. Incardona, and I. F. Sbalzarini, "A c++ expression system for partial differential equations enables generic simulations of biological hydrodynamics," *European Physical Journal E*, vol. 44, 9 2021.

[144] P. G. de Gennes and J. Prost, *The physics of liquid crystals*. International Series of Monographs on Physics, Oxford, England: Clarendon Press, 2 ed., Apr. 1995.

[145] J. D. Eldredge, A. Leonard, and T. Colonius, "A General Deterministic Treatment of Derivatives in Particle Methods," *Journal of Computational Physics*, vol. 180, pp. 686–709, Aug. 2002.

[146] G. Bourantas, B. L. Cheeseman, R. Ramaswamy, and I. F. Sbalzarini, "Using dc pse operator discretization in eulerian meshless collocation methods improves their robustness in complex geometries," *Computers and Fluids*, vol. 136, pp. 285–300, 9 2016.

[147] J. H. Ferziger and M. Perić, *Computational Methods for Fluid Dynamics*. Springer Berlin Heidelberg, 2002.

[148] G.-H. Cottet, J.-M. Etancelin, F. Perignon, and C. Picard, "High order semi-Lagrangian particle methods for transport equations: numerical analysis and implementation issues," *ESAIM: M2AN*, vol. 48, pp. 1029–1060, July 2014. Number: 4 Publisher: EDP Sciences.

[149] A. Magni and G.-H. Cottet, "Accurate, non-oscillatory, remeshing schemes for particle methods," *Journal of Computational Physics*, vol. 231, pp. 152–172, Jan. 2012.

[150] J. J. Monaghan, "Extrapolating B splines for interpolation," *J. Comput. Phys.*, vol. 60, pp. 253–262, 1985.

[151] R. Ramaswamy, G. Bourantas, F. Jülicher, and I. F. Sbalzarini, "A hybrid particle-mesh method for incompressible active polar viscous gels," *Journal of Computational Physics*, vol. 291, pp. 334–361, 6 2015.

[152] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

[153] R. G. Barrera, G. A. Estevez, and J. Giraldo, "Vector spherical harmonics and their application to magnetostatics," *Eur. J. Phys.*, vol. 6, pp. 287–294, Oct. 1985. Publisher: IOP Publishing.

[154] I. Nitschke, S. Reuther, and A. Voigt, "Liquid Crystals on Deformable Surfaces," *arXiv:1911.11859 [cond-mat]*, Nov. 2019. arXiv: 1911.11859.

[155] R. Wang, Z. Yang, L. Liu, and Q. Chen, "Discretizing Laplace–Beltrami Operator from Differential Quantities," *Commun. Math. Stat.*, vol. 1, pp. 331–350, Sept. 2013.

[156] S. Chun, "Method of moving frames to solve conservation laws on curved surfaces," *Journal of Scientific Computing*, vol. 53, no. 2, pp. 268–294, 2012.

[157] S. Chun and C. Eskilsson, "Method of moving frames to solve the shallow water equations on arbitrary rotating curved surfaces," *Journal of Computational Physics*, vol. 333, pp. 1–23, 2017.

[158] E. Bachini, M. W. Farthing, and M. Putti, "Intrinsic finite element method for advection-diffusion-reaction equations on surfaces," *Journal of Computational Physics*, vol. 424, p. 109827, Jan. 2021.

[159] J. Grande, M. Olshanskii, and A. Reusken, "A space-time fem for pdes on evolving surfaces," pp. 211–222, 2014.

[160] I. Nitschke, S. Reuther, and A. Voigt, "Discrete Exterior Calculus (DEC) for the Surface Navier-Stokes Equation," in *Transport Processes at Fluidic Interfaces* (D. Bothe and A. Reusken, eds.), pp. 177–197, Cham: Springer International Publishing, 2017.

[161] S. Leung and H. Zhao, "A grid based particle method for moving interface problems," *J. Comput. Phys.*, vol. 228, no. 8, pp. 2993–3024, 2009.

[162] M. Olshanskii and A. Reusken, "Trace finite element methods for PDEs on surfaces," *Lecture Notes in Computational Science and Engineering*, vol. 121, pp. 211–258, 2017.

[163] G.-H. Cottet and E. Maitre, "A semi-implicit level set method for multiphase flows and fluid–structure interaction problems," *Journal of Computational Physics*, vol. 314, pp. 80–92, June 2016.

[164] M. Bergdorf, I. F. Sbalzarini, and P. Koumoutsakos, "A Lagrangian particle method for reaction–diffusion systems on deforming surfaces," *J. Math. Biol.*, vol. 61, pp. 649–663, Nov. 2010.

[165] S. J. Ruuth and B. Merriman, "A simple embedding method for solving partial differential equations on surfaces," *Journal of Computational Physics*, vol. 227, pp. 1943–1961, Jan. 2008.

[166] C. B. Macdonald, J. Brandman, and S. J. Ruuth, "Solving eigenvalue problems on curved surfaces using the Closest Point Method," *Journal of Computational Physics*, vol. 230, pp. 7944–7956, Sept. 2011. arXiv: 1106.4351.

[167] A. J. James and J. Lowengrub, "A surfactant-conserving volume-of-fluid method for interfacial flows with insoluble surfactant," *J. Comput. Phys.*, vol. 201, pp. 685–722, 2004.

[168] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-based fast local level set method," *J. Comput. Phys.*, vol. 155, pp. 410–438, 1999.

[169] T. März and C. B. Macdonald, "Calculus on surfaces with general closest point functions," *SIAM Journal on Numerical Analysis*, vol. 50, no. 6, pp. 3303–3328, 2012.

[170] Á. González, "Measurement of areas on a sphere using Fibonacci and latitude–longitude lattices," *Mathematical Geosciences*, vol. 42, pp. 49–64, Nov. 2009.

[171] A. Singh, Q. Vagne, F. Jülicher, and I. F. Sbalzarini, "Spontaneous flow instabilities of active polar fluids in three dimensions," *Phys. Rev. Res.*, vol. 5, p. L022061, Jun 2023.

[172] S. Shankar, A. Souslov, M. J. Bowick, M. C. Marchetti, and V. Vitelli, "Topological active matter," *Nat Rev Phys*, pp. 1–19, May 2022. Publisher: Nature Publishing Group.

[173] S. Shankar and M. C. Marchetti, "Hydrodynamics of Active Defects: From Order to Chaos to Defect Ordering," *Phys. Rev. X*, vol. 9, p. 041047, Dec. 2019.

[174] J. M. Kosterlitz and D. J. Thouless, "Ordering, metastability and phase transitions in two-dimensional systems," *Journal of Physics C: Solid State Physics*, vol. 6, pp. 1181–1203, Apr. 1973.

[175] N. Astrakhantsev, T. Westerhout, A. Tiwari, K. Choo, A. Chen, M. H. Fischer, G. Carleo, and T. Neupert, "Broken-symmetry ground states of the heisenberg model on the pyrochlore lattice," *Phys. Rev. X*, vol. 11, p. 041021, Oct 2021.

[176] F. Jülicher, K. Kruse, J. Prost, and J. F. Joanny, "Active behavior of the Cytoskeleton," *Physics Reports*, vol. 449, pp. 3–28, Sept. 2007.

[177] A. C. Martin, "Pulsation and stabilization: Contractile forces that underlie morphogenesis," *Developmental Biology*, vol. 341, pp. 114–125, May 2010.

[178] S. Shankar, A. Souslov, M. J. Bowick, M. C. Marchetti, and V. Vitelli, "Topological active matter," *arXiv:2010.00364 [cond-mat]*, Oct. 2020. arXiv: 2010.00364.

[179] P. Guillamat, C. Blanch-Mercader, G. Pernollet, K. Kruse, and A. Roux, "Integer topological defects organize stresses driving tissue morphogenesis," *Nat. Mater.*, Feb. 2022.

[180] D. Saintillan and M. J. Shelley, "Instabilities and pattern formation in active particle suspensions: Kinetic theory and continuum simulations," *Phys. Rev. Lett.*, vol. 100, p. 178103, Apr 2008.

[181] D. Saintillan and M. J. Shelley, "Instabilities, pattern formation, and mixing in active suspensions," *Physics of Fluids*, vol. 20, no. 12, p. 123304, 2008.

[182] R. Aditi Simha and S. Ramaswamy, "Hydrodynamic fluctuations and instabilities in ordered suspensions of self-propelled particles," *Phys. Rev. Lett.*, vol. 89, p. 058101, Jul 2002.

[183] G. Subramanian and D. L. Koch, "Critical bacterial concentration for the onset of collective swimming," *Journal of Fluid Mechanics*, vol. 632, pp. 359–400, Aug 2009.

[184] F. G. Woodhouse and R. E. Goldstein, "Spontaneous circulation of confined active suspensions," *Phys. Rev. Lett.*, vol. 109, p. 168105, Oct 2012.

[185] V. Fréedericksz and V. Zolina, "Forces causing the orientation of an anisotropic liquid," *Trans. Faraday Soc.*, vol. 29, pp. 919–930, Jan. 1933. Publisher: The Royal Society of Chemistry.

[186] S. Shoarinejad and M. Shahzamanian, "On the numerical study of Frederick transition in nematic liquid crystals," *Journal of Molecular Liquids*, vol. 138, pp. 14–19, Feb. 2008.

[187] G. Duclos, C. Blanch-Mercader, V. Yashunsky, G. Salbreux, J.-F. Joanny, J. Prost, and P. Silberzan, "Spontaneous shear flow in confined cellular nematics," *Nature Phys*, vol. 14, pp. 728–732, July 2018. Number: 7 Publisher: Nature Publishing Group.

[188] S. Ramaswamy, "Active fluids," *Nature Reviews Physics*, vol. 1, pp. 640–642, Nov. 2019.

[189] F. G. Woodhouse and R. E. Goldstein, "Spontaneous circulation of confined active suspensions," *Phys. Rev. Lett.*, vol. 109, p. 168105, Oct 2012.

[190] H. Wioland, F. G. Woodhouse, J. Dunkel, J. O. Kessler, and R. E. Goldstein, "Confinement stabilizes a bacterial suspension into a spiral vortex," *Phys. Rev. Lett.*, vol. 110, p. 268102, Jun 2013.

[191] G. Sarfati, A. Maitra, R. Voituriez, J. C. Galas, and A. Estevez-Torres, "Crosslinking and depletion determine spatial instabilities in cytoskeletal active matter," *Soft Matter*, vol. 18, pp. 3793–3800, 4 2022.

[192] S. Chandragiri, A. Doostmohammadi, J. M. Yeomans, and S. P. Thampi, "Flow states and transitions of an active nematic in a three-dimensional channel," *Physical Review Letters*, vol. 125, 9 2020.

[193] M. R. Nejad and J. M. Yeomans, "Active extensile stress promotes 3d director orientations and flows," *Phys. Rev. Lett.*, vol. 128, p. 048001, Jan 2022.

[194] P. Suhrcke, "Active turbulence in extensile polar fluids - a step into the third dimension," 2022.

[195] Y. Pomeau and P. Manneville, "Intermittent transition to turbulence in dissipative dynamical systems," *Communications in Mathematical Physics*, vol. 74, pp. 189–197, June 1980.

[196] H. Chate, "Spatiotemporal intermittency regimes of the one-dimensional complex ginzburg-landau equation," *Nonlinearity*, vol. 7, p. 185, jan 1994.

[197] R. Alert, J. Casademunt, and J.-F. Joanny, "Active Turbulence," *arXiv:2104.02122 [cond-mat, physics:nlin, physics:physics]*, Apr. 2021. arXiv: 2104.02122.

[198] G. Benettin, L. Galgani, A. Giorgilli, and J.-M. Strelcyn, "Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. part 2: Numerical method," *Meccanica*, vol. 15, no. 1, pp. 21–30, 1980.

[199] C. Skokos, "The lyapunov characteristic exponents and their computation," in *Dynamics of Small Solar System Bodies and Exoplanets*, pp. 63–135, Springer, 2010.

[200] D. Saintillan, M. J. Shelley, and A. Zidovska, "Extensile motor activity drives coherent motions in a model of interphase chromatin," *Proceedings of the National Academy of Sciences*, vol. 115, no. 45, pp. 11442–11447, 2018.

[201] A. Mahajan, W. Yan, A. Zidovska, D. Saintillan, and M. J. Shelley, "Euchromatin activity enhances segregation and compaction of heterochromatin in the cell nucleus," *Phys. Rev. X*, vol. 12, p. 041033, Dec 2022.

[202] S. Weady, D. B. Stein, A. Zidovska, and M. J. Shelley, "Conformations, correlations, and instabilities of a flexible fiber in an active fluid," *arXiv preprint arXiv:2309.12225*, 2023.

[203] A. Opathalage, M. M. Norton, M. P. N. Juniper, B. Langeslay, S. A. Aghvami, S. Fraden, and Z. Dogic, "Self-organized dynamics and the transition to turbulence of confined active nematics," *Proceedings of the National Academy of Sciences*, vol. 116, no. 11, pp. 4788–4797, 2019.

[204] J. Hardoüin, J. Laurent, T. Lopez-Leon, J. Ignés-Mullol, and F. Sagués, "Active microfluidic transport in two-dimensional handlebodies," *Soft Matter*, vol. 16, no. 40, pp. 9230–9241, 2020.

[205] J. Hardoüin, C. Doré, J. Laurent, T. Lopez-Leon, J. Ignés-Mullol, and F. Sagués, "Active boundary layers in confined active nematics," *Nature Communications*, vol. 13, no. 1, p. 6675, 2022.

[206] C. Joshi, Z. Zarei, M. M. Norton, S. Fraden, A. Baskaran, and M. F. Hagan, "From disks to channels: Dynamics of active nematics confined to an annulus," 2023.

[207] M. Neef and K. Kruse, "Generation of stationary and moving vortices in active polar fluids in the planar taylor-couette geometry," *Physical Review E*, vol. 90, no. 5, p. 052703, 2014.

[208] S. Chen, P. Gao, and T. Gao, "Dynamics and structure of an apolar active suspension in an annulus," *Journal of Fluid Mechanics*, vol. 835, pp. 393–405, 2018.

[209] T. N. Shendruk, A. Doostmohammadi, K. Thijssen, and J. M. Yeomans, "Dancing disclinations in confined active nematics," *Soft Matter*, vol. 13, no. 21, pp. 3853–3862, 2017.

[210] K.-T. Wu, J. B. Hishamunda, D. T. N. Chen, S. J. DeCamp, Y.-W. Chang, A. Fernández-Nieves, S. Fraden, and Z. Dogic, "Transition from turbulent to coherent flows in confined three-dimensional active fluids," *Science*, vol. 355, no. 6331, p. eaal1979, 2017.

[211] B. O. Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[212] J. Zhao, C. Duclut, A. Singh, R. Golipour, A. Pham, B. Golshaei, C. Guan, M. Li, R. Oldenbourg, S. W. Grill, I. F. Sbalzarini, J. L. Harden, F. Jülicher, and C. F. Schmidt, "Active force driven large-scale directional flow of isotropic cytoskeletal networks," In preparation.

[213] W. R. Inc., "Mathematica, Version 13.3." Champaign, IL, 2023.

# Appendices

# Appendix A

# Expression templates

Expression templates are a technique in C++ that allows us to manipulate mathematical expressions in a way that is efficient and flexible [108]. This approach is often used in libraries for scientific computing to express code in a simplified way, while retaining performance [98, 106, 109, 126].

## A.1 Example: vectorial expressions

Consider the operation `z = x + y`, where `x`, `y`, and `z` are vectors. Without expression templates, this operation may involve creating an intermediate temporary vector to store the result of `x + y` before assigning it to `z`, which can be expensive in terms of memory and performance. With expression templates, we can defer the actual calculations until the assignment, and then calculate the result element by element, without creating a temporary vector. For example,

```
template<typename E>
class VecExpression {
  public:
  float operator[](size_t i) const { return static_cast<E const&>(*this)[i]; }
  size_t size() const { return static_cast<E const&>(*this).size(); }
};

class Vec : public VecExpression<Vec> {
  // ...
};

template<typename E1, typename E2>
class VecSum : public VecExpression<VecSum<E1, E2>> {
  // ...
};

template<typename E1, typename E2>
```

```
18  VecSum<E1,E2> operator+(E1 const& u, E2 const& v) {
19    return VecSum<E1, E2>(u, v);
20  }
21
22  template<typename E>
23  Vec& Vec::operator=(VecExpression<E> const& vec) {
24    // ...
25  }
```

Listing A.1: C++ code with expression templates

The above code demonstrates the use of expression templates in C++.

- `template<typename E> class VecExpression {...}`: This is the base class for all kinds of vector expressions. It is a template class that takes one type parameter `E`. It contains two functions `operator[]` and `size()`, which are common to all vector expressions. These functions are defined in terms of the same functions of the derived class, and the derived class is accessed through `static_cast`.

- `class Vec : public VecExpression<Vec> {...}`: `Vec` is the main vector class that other vector expressions operate on. It is derived from `VecExpression`, with `Vec` itself as the template argument. This pattern is known as the Curiously Recurring Template Pattern (CRTP).

- `template<typename E1, typename E2> class VecSum : public VecExpression<VecSum<E1, E2>> {...}`:
  This is a template class for representing the sum of two vector expressions. It is derived from `VecExpression`, with `VecSum` itself as the template argument (another example of CRTP).

- `template<typename E1, typename E2> VecSum<E1,E2> operator+(E1 const& u, E2 const& v) {...}`: This is the function that is called when the `+` operator is used with two vector expressions. It returns a `VecSum` object representing the sum of the expressions.

- `template<typename E> Vec& Vec::operator=(VecExpression<E> const& vec) {...}`: This is the assignment operator for the `Vec` class. When an expression is assigned to a `Vec` object, this function is called. The function should loop over the elements of the `vec` argument and assign the values to the `Vec` object, but the implementation details have been left out in the example.

The key idea of expression templates is that when an expression is constructed (e.g., `x + y`), no actual computations are performed. Instead, a new expression object is created (in this case, a `VecSum` object), which contains references to the operands and remembers the operation to be performed. The actual computations are deferred until the expression is

assigned to a `Vec` object, at which point the assignment operator of the `Vec` class performs the computations element by element, without creating a temporary vector.

This approach eliminates the need for temporary vectors and can lead to significant performance improvements for large vectors. However, the code can be quite complex and difficult to understand, and requires familiarity with template metaprogramming in C++ to comprehend error messages.

## A.2 Example: DC-PSE differential operator expressions

Next, we describe a code example used for creating differential operator expressions in OpenFPM.

```
//Addition operation node
template<typename exp1, typename DCPSE_type>
class vector_dist_expression_op<exp1, DCPSE_type, VECT_DCPSE_V_SUM> {
  const exp1 o1;
  DCPSE_type (&dcp)[DCPSE_type::vtype::dims];
  // More members
  static const int dims = DCPSE_type::vtype::dims;
  typedef typename DCPSE_type::vtype::stype stype;

  template<typename r_type= typename std::remove_reference<decltype(o1.value(vect_dist_key_dx(0)))
      >::type>
  inline r_type value(const vect_dist_key_dx &key) const {
    //code to evaluate the operator
  }

  template<typename Sys_eqs, typename pmap_type, typename unordered_map_type, typename
      coeff_type>
  inline void value_nz(pmap_type &p_map, const vect_dist_key_dx &key, unordered_map_type &cols,
      coeff_type &coeff,
  unsigned int comp) const {
    //code to update non-zero stencil coeffiecients based on the operator
      }
  vtype &getVector() {
    return o1.getVector();
  }
};

//Differential operator node
template<template<unsigned int, typename, typename...> class Dcpse_type = Dcpse>
class Derivative_x_T {
  void *dcpse;
  // More methods
  template<typename particles_type>
  void deallocate(particles_type &parts) {
    delete (Dcpse_type<particles_type::dims, particles_type> *) dcpse;
  }
```

```
34   template<typename particles_type>
35   void update(particles_type &particles) {
36     auto dcpse_temp = (Dcpse_type<particles_type::dims, particles_type> *) dcpse;
37     dcpse_temp->initializeUpdate(particles);
38   }
39 };
```

Listing A.2: Expression templates for differential operators in OpenFPM

In this listing A.2, we illustrate the application of expression templates to DC-PSE operators in the context of expression templates. We define an operation that represents the differential operator `Dx` applied to an expression `P` in section 2.1, computations are deferred until necessary, and `Dx(P)` represents the operation rather than computing the derivative. The key elements are:

- `vector_dist_expression_op<exp1, DCPSE_type, VECT_DCPSE_V_SUM>`:
  This class template represents the operation of applying a differential operator to an expression. The template parameters are the expression type (`exp1`), the differential operator type (`DCPSE_type`), and an operation tag (`VECT_DCPSE_V_SUM`). We store the expression on which the differential operator is applied in `exp1 o1`, and the differential operator itself in `DCPSE_type (&dcp)[DCPSE_type::vtype::dims]`. When we call the `value(const vect_dist_key_dx &key) const` method, it computes the derivative at a given key, while the `value_nz(...) const` method updates the stencil coefficient values in the unordered map.

- `value(const vect_dist_key_dx &key)`: This function returns the result of the differential operator applied on the given expression, for a given key. The function operates by iterating over each dimension, computing the differential operator and summing them up. The key argument specifies the particle for which the computation is being done.

- `value_nz(pmap_type &p_map, const vect_dist_key_dx &key, unordered_map_type &cols, coeff_type &coeff, unsigned int comp)`: This function is used for the computation of the sparse system for the linear solver. It calculates the non-zero elements of the matrix corresponding to the operation in the expression template. For a given key, the function computes the contribution of all its neighboring particles (non-zero entries in the sparse matrix) and stores them in a map, indexed by the column number in the system. The p_map is used to map the particle keys to a global index for the linear solver, and cols is the map that stores the coefficients of the matrix. coeff is the scaling factor for the computation, and comp denotes the component of the vector if the property is a vector.

- `Derivative_x_T`:
  This class serves as a factory for creating instances of `vector_dist_expression_op`.

It encapsulates the details of constructing a differential operator and applying it to an expression. Our operator function, `operator()(operand_type arg)`, generates an instance of `vector_dist_expression_op` that signifies the derivative of the provided argument. Other member functions of `Derivative_x_T` supports deallocating resources, visualizing the kernel, validating moment conditions, saving and loading the operator, and updating the operator.

- `getVector()` and `getVector() const` methods: These methods return the vector on which the operator is acting. It is important in some cases to access the original vector that is used in the expression.

- `deallocate(particles_type &parts)`: This method in `Derivative_x_T` class template deallocates the memory associated with the operator. It is critical to manage memory in a way that prevents memory leaks and ensures efficient use of resources.

- `update(particles_type &particles)`: This method updates the DCPSE operator by recomputing the DCPSE Kernels. It is essential when the particles move, and the operator needs to be updated to reflect the changes in the coefficients based on the new particle positions.

When creating an expression like `Dx(P)`, it does not compute the derivative but builds an object symbolizing the derivative. The actual calculation of the derivative is postponed until the `value` method of `vector_dist_expression_op` is invoked. When we call `value(key)` on this object, it computes the derivative at the provided key. This approach can offer significant performance enhancements by evading unnecessary computations and avoiding the creation of temporary objects. Again, we highlight the advantage of deferred computation using expression templates. The actual derivative computation only happens when the `value` method of the `vector_dist_expression_op` object is invoked, leading to efficient computations.

# Appendix B

# Stokes flow equations of the Ericksen-Leslie active hydrodynamics model

## B.1 Two-dimensional equations

The passive stress in an active fluid, $\sigma^{(\mathrm{p})}_{\alpha\beta} = \sigma^{(\mathrm{s})}_{\alpha\beta} + \sigma^{(\mathrm{e})}_{\alpha\beta} + \sigma^{(\mathrm{ant})}_{\alpha\beta}$ is decomposed as the sum of the symmetric (s), antisymmetric (ant), and equilibrium (e) stresses. The equilibrium stress, also called the Ericksen stress, is given by

$$\sigma^{(\mathrm{e})}_{\alpha\beta} = -\frac{\partial f}{\partial\left(\partial_\beta p_\gamma\right)}\partial_\alpha p_\gamma, \tag{B.1}$$

where $f$ is free energy functional from Eq. (3.3). The symmetric stress is the deviatoric part of the symmetric stress tensor (i.e., the total passive symmetric stress). The anti-symmetric stress is given by,

$$\sigma^{(\mathrm{ant})}_{\alpha\beta} = \frac{1}{2}\left(p_\alpha h_\beta - p_\beta h_\alpha\right). \tag{B.2}$$

Following Eq. (3.1a), the molecular field $h_\alpha$ is decomposed into the component $h_\parallel$ parallel and the component $h_\perp$ perpendicular to the local polarity $\mathbf{p}$, where $h_\parallel$ is derived as a Lagrange multiplier,

$$h_\parallel = -\gamma\left[\lambda\Delta\mu - \nu\frac{u_{\mathrm{xx}}p_{\mathrm{x}}^2}{p_{\mathrm{x}}^2+p_{\mathrm{y}}^2} - \nu\frac{u_{\mathrm{yy}}p_{\mathrm{y}}^2}{p_{\mathrm{x}}^2+p_{\mathrm{y}}^2} - 2\nu\frac{u_{\mathrm{xy}}p_{\mathrm{x}}p_{\mathrm{y}}}{p_{\mathrm{x}}^2+p_{\mathrm{y}}^2}\right] \tag{B.3}$$

to enforce $\|\mathbf{p}\| = 1$. Equations (3.1b) and (3.1c) are then combined to the final component-wise expressions [17]:

$$\eta\partial_x^2 v_{\mathrm{x}} + \eta\partial_y^2 v_{\mathrm{x}} - \partial_x\Pi + \frac{\nu}{2}\partial_x\left[\frac{\gamma\nu u_{\mathrm{xx}} p_{\mathrm{x}}^2\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] +$$
$$\frac{\nu}{2}\partial_x\left[\frac{2\gamma\nu u_{\mathrm{xy}} p_{\mathrm{x}} p_{\mathrm{y}}\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] + \frac{\nu}{2}\partial_x\left[\frac{\gamma\nu u_{\mathrm{yy}} p_{\mathrm{y}}^2\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] +$$
$$\frac{\nu}{2}\partial_y\left[\frac{2\gamma\nu u_{\mathrm{xx}} p_{\mathrm{x}}^3 p_{\mathrm{y}}}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] + \frac{\nu}{2}\partial_y\left[\frac{4\gamma\nu u_{\mathrm{xy}} p_{\mathrm{x}}^2 p_{\mathrm{y}}^2}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] + \frac{\nu}{2}\partial_y\left[\frac{2\gamma\nu u_{\mathrm{yy}} p_{\mathrm{x}} p_{\mathrm{y}}^3}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right]$$
$$= -\frac{1}{2}\partial_y\left(h_\perp\right) + \zeta\partial_x\left(\Delta\mu p_{\mathrm{x}}^2\right) + \zeta\partial_y\left(\Delta\mu p_{\mathrm{x}} p_{\mathrm{y}}\right) -$$
$$\zeta\partial_x\left(\Delta\mu\frac{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}{2}\right) - \frac{\nu}{2}\partial_x\left(-2h_\perp p_{\mathrm{x}} p_{\mathrm{y}}\right) -$$
$$\frac{\nu}{2}\partial_y\left[h_\perp\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)\right] - \partial_x\sigma_{\mathrm{xx}}^{(\mathrm{e})} - \partial_y\sigma_{\mathrm{xy}}^{(\mathrm{e})} +$$
$$\frac{\nu}{2}\partial_x\left[\gamma\lambda\Delta\mu\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)\right] - \frac{\nu}{2}\partial_y\left(-2\gamma\lambda\Delta\mu p_{\mathrm{x}} p_{\mathrm{y}}\right), \quad \text{(B.4)}$$

$$\eta\partial_x^2 v_{\mathrm{y}} + \eta\partial_y^2 v_{\mathrm{y}} - \partial_y\Pi + \frac{\nu}{2}\partial_y\left[\frac{-\gamma\nu u_{\mathrm{xx}} p_{\mathrm{x}}^2\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] +$$
$$\frac{\nu}{2}\partial_y\left[\frac{-2\gamma\nu u_{\mathrm{xy}} p_{\mathrm{x}} p_{\mathrm{y}}\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] + \frac{\nu}{2}\partial_y\left[\frac{-\gamma\nu u_{\mathrm{yy}} p_{\mathrm{y}}^2\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] +$$
$$\frac{\nu}{2}\partial_x\left[\frac{2\gamma\nu u_{\mathrm{x}x} p_{\mathrm{x}}^3 p_{\mathrm{y}}}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] + \frac{\nu}{2}\partial_x\left[\frac{4\gamma\nu u_{\mathrm{xx}} p_{\mathrm{x}}^2 p_{\mathrm{y}}^2}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right] + \frac{\nu}{2}\partial_x\left[\frac{2\gamma\nu u_{\mathrm{yy}} p_{\mathrm{x}} p_{\mathrm{y}}^3}{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}\right]$$
$$= -\frac{1}{2}\partial_x\left(-h_\perp\right) + \zeta\partial_y\left(\Delta\mu p_{\mathrm{y}}^2\right) + \zeta\partial_x\left(\Delta\mu p_{\mathrm{x}} p_{\mathrm{y}}\right) -$$
$$\zeta\partial_y\left(\Delta\mu\frac{p_{\mathrm{x}}^2 + p_{\mathrm{y}}^2}{2}\right) - \frac{\nu}{2}\partial_y\left(2h_\perp p_{\mathrm{x}} p_{\mathrm{y}}\right) -$$
$$\frac{\nu}{2}\partial_x\left[h_\perp\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)\right] - \partial_x\sigma_{\mathrm{yx}}^{(\mathrm{e})} - \partial_y\sigma_{\mathrm{yy}}^{(\mathrm{e})} -$$
$$\frac{\nu}{2}\partial_y\left[\gamma\lambda\Delta\mu\left(p_{\mathrm{x}}^2 - p_{\mathrm{y}}^2\right)\right] - \frac{\nu}{2}\partial_x\left(-2\gamma\lambda\Delta\mu p_{\mathrm{x}} p_{\mathrm{y}}\right). \quad \text{(B.5)}$$

Together with the incompressibility condition in Eq. (3.1b), this completes the force balance in two dimensions.

## B.2 Three-dimensional equations

We decompose the molecular field $\mathbf{h}$ into parallel and perpendicular components,

$$h_{\parallel} = \mathbf{p} \cdot \mathbf{h} = p_x h_x + p_y h_y + p_z h_z \tag{B.6a}$$

$$\mathbf{h}_{\perp} = \mathbf{p} \times \mathbf{h} = (h_{\perp x}, h_{\perp y}, h_{\perp z})$$

$$= (p_y h_z - p_z h_y, p_z h_x - p_x h_z, p_x h_y - p_y h_x). \tag{B.6b}$$

The vector $\mathbf{h}_{\perp}$ is computed from the variational derivative of the Frank free energy density

$$F_{3D} = \frac{K_s}{2}(\nabla \cdot \mathbf{p})^2 + \frac{K_t}{2}(\mathbf{p} \cdot \nabla \times \mathbf{p})^2 + \frac{K_b}{2}(\mathbf{p} \times (\nabla \times \mathbf{p}))^2 - \frac{1}{2}h_{\parallel}^0 \|\mathbf{p}\|^2 \tag{B.7}$$

with respect to $\mathbf{p}$. The total stress $\sigma_{\alpha\beta}^{(\mathrm{tot})} = \sigma_{\alpha\beta}^{(\mathrm{s})} + \sigma_{\alpha\beta}^{(\mathrm{ant})} + \sigma_{\alpha\beta}^{(\mathrm{e})}$ is decomposed as the sum of the symmetric (s), antisymmetric (ant), and equilibrium (e) stresses. The equilibrium stress, also called the Ericksen stress, is given by

$$\sigma_{\alpha\beta}^{(\mathrm{e})} = -\frac{\partial F_{3D}}{\partial \left(\partial_\beta p_\gamma\right)} \partial_\alpha p_\gamma, \tag{B.8}$$

with $F_{3D}$ from Eq. (B.7). The anti-symmetric stress is

$$\sigma_{\alpha\beta}^{(\mathrm{ant})} = \frac{1}{2}\left(p_\alpha h_\beta - p_\beta h_\alpha\right). \tag{B.9}$$

Setting $p_\gamma \frac{Dp_\gamma}{Dt} = 0$ to maintain constant polarity magnitude $p_\gamma p_\gamma$, we derive the Lagrange multiplier

$$h_{\parallel} = -\gamma\Big[\lambda\Delta\mu - \frac{\nu}{p_x^2 + p_y^2 + p_z^2}\Big(u_{xx}p_x^2 + u_{yy}p_y^2 + u_{zz}p_z^2 + 2u_{xy}p_x p_y + 2u_{yz}p_y p_z + 2u_{xz}p_x p_z\Big)\Big]. \tag{B.10}$$

Substituting the decomposition of $\mathbf{h}$

$$h_x = h_{\parallel}p_x - h_{\perp z}p_y + h_{\perp z}p_y \tag{B.11a}$$

$$h_y = h_{\parallel}p_y + h_{\perp z}p_x - h_{\perp x}p_z \tag{B.11b}$$

$$h_z = h_{\parallel}p_z + h_{\perp x}p_y - h_{\perp y}p_x \tag{B.11c}$$

Similar to two dimensions, three dimensional force balance is derived using *Mathematica* [213] as,

x-component:

$$\eta(2\partial_{xx}v_x + \partial_{xy}v_y + \partial_{yy}v_x + \partial_{xz}v_z + \partial_{zz}v_x) + \nu^2\gamma\Bigg[$$

$$\frac{1}{(p_x^2+p_y^2+p_z^2)^2}\Big[-2p_xp_y(p_x\partial_yp_x+p_y\partial_yp_y+p_z\partial_yp_z)(p_x^2\partial_xv_x+p_xp_y(\partial_xv_y+\partial_yv_x)+p_y^2\partial_yv_y$$
$$+p_xp_z(\partial_xv_z+\partial_zv_x)+p_yp_z(\partial_yv_z+\partial_zv_y)+p_z^2\partial_zv_z)-2p_xp_z(p_x\partial_zp_x+p_y\partial_zp_y$$
$$+p_z\partial_zp_z)(p_x^2\partial_xv_x+p_xp_y(\partial_xv_y+\partial_yv_x)+p_y^2\partial_yv_y+p_xp_z(\partial_xv_z+\partial_zv_x)$$
$$+p_yp_z(\partial_yv_z+\partial_zv_y)+p_z^2\partial_zv_z)+\frac{1}{3}(2(p_x\partial_xp_x+p_y\partial_xp_y+p_z\partial_xp_z)$$
$$(-2p_x^2+p_y^2+p_z^2)(p_x^2\partial_xv_x+p_xp_y(\partial_xv_y+\partial_yv_x)+p_y^2\partial_yv_y+p_xp_z(\partial_xv_z+\partial_zv_x)$$
$$+p_yp_z(\partial_yv_z+\partial_zv_y)+p_zp_z\partial_zv_z))\Big]$$
$$+\frac{1}{p_x^2+p_y^2+p_z^2}\Big[(\frac{4p_x\partial_xp_x}{3}+p_x\partial_yp_y+p_x\partial_zp_z-\frac{2p_y\partial_xp_y}{3}+p_y\partial_yp_x-\frac{2p_z\partial_xp_z}{3}+p_z\partial_zp_x)$$
$$(p_x^2\partial_xv_x+p_xp_y(\partial_xv_y+\partial_yv_x)+p_y^2\partial_yv_y+p_xp_z(\partial_xv_z\partial_zv_x)+p_yp_z(\partial_yv_z+\partial_zv_y)$$
$$+p_z^2\partial_zv_z)+\frac{1}{3}((2p_x^2-p_y^2-p_z^2)(2p_x\partial_xp_x\partial_xv_x+p_x^2\partial_{xx}v_x+p_y^2\partial_{xx}v_y+p_z^2\partial_{xx}v_z$$
$$+p_x\partial_xp_y(\partial_xv_y+\partial_yv_x)+p_y\partial_xp_x(\partial_xv_y+\partial_yv_x)+2p_y\partial_xp_y\partial_yv_y+p_xp_y(\partial_{xx}v_y$$
$$+\partial_{xy}v_x)+p_x\partial_xp_z(\partial_xv_z+\partial_zv_x)+p_z\partial_xp_x(\partial_xv_z+\partial_zv_x)+p_y\partial_xp_z(\partial_yv_z+\partial_zv_y)$$
$$+p_z\partial_xp_y(\partial_yv_z+\partial_zv_y)+2p_z\partial_xp_z\partial_zv_z+p_xp_z(\partial_{xx}v_z+\partial_{xz}v_x)+p_yp_z(\partial_{xy}v_z$$
$$+\partial_{xz}v_y)))+p_xp_y(2p_x\partial_yp_x\partial_xv_x+p_x^2\partial_{xy}v_x+p_x\partial_yp_y(\partial_xv_y+\partial_yv_x)$$
$$+p_y\partial_yp_x(\partial_xv_y+\partial_yv_x)+2p_y\partial_yp_y\partial_yv_y+p_xp_y(\partial_{xy}v_y+\partial_{yy}v_x)+p_y^2\partial_{yy}v_y$$
$$+p_z^2\partial_{yz}v_z+p_x\partial_yp_z(\partial_xv_z+\partial_zv_x)+p_z\partial_yp_x(\partial_xv_z+\partial_zv_x)+p_y\partial_yp_z(\partial_yv_z$$
$$+\partial_zv_y)+p_z\partial_yp_y(\partial_yv_z\partial_zv_y)+2p_z\partial_yp_z\partial_zv_z+p_xp_z(\partial_{xy}v_z+\partial_{zy}v_x)$$
$$+p_yp_z(\partial_{yy}v_z+\partial_{zy}v_y))+p_xp_z(2p_x\partial_zp_x\partial_xv_x+p_x^2\partial_{xz}v_x+p_x\partial_zp_y(\partial_xv_y+\partial_yv_x)$$
$$+p_y\partial_zp_x(\partial_xv_y+\partial_yv_x)+2p_y\partial_zp_y\partial_yv_y+p_xp_y(\partial_{xz}v_y+\partial_{yz}v_x)+p_y^2\partial_{yz}v_y$$
$$+p_x\partial_zp_z(\partial_xv_z+\partial_zv_x)+p_z\partial_zp_x(\partial_xv_z+\partial_zv_x)+p_y\partial_zp_z(\partial_yv_z+\partial_zv_y)$$
$$+p_z\partial_zp_y(\partial_yv_z+\partial_zv_y)+2p_z\partial_zp_z\partial_zv_z+p_xp_z(\partial_{xz}v_z+\partial_{zz}v_x)+p_yp_z(\partial_{yz}v_z$$
$$+\partial_{zz}v_y)+p_z^2\partial_{zz}v_z)\Big]\Big]$$

$$=-\Big[-0.5p_xh_{\perp,y}\partial_zp_x+0.5p_xh_{\perp,z}\partial_yp_x+0.5p_yh_{\perp,x}\partial_zp_x+0.5p_yh_{\perp,z}\partial_yp_y+0.5p_yh_{\perp,z}\partial_zp_z$$
$$-0.5p_x(-h_{\perp,x}\partial_zp_y+h_{\perp,y}\partial_zp_x+p_x\partial_zh_{\perp,y}-p_y\partial_zh_{\perp,x})-0.5p_zh_{\perp,x}\partial_yp_x-0.5p_zh_{\perp,y}\partial_yp_y$$
$$-0.5p_zh_{\perp,y}\partial_zp_z+0.5p_x(-h_{\perp,x}\partial_yp_z+h_{\perp,z}\partial_yp_x+p_x\partial_yh_{\perp,z}-p_z\partial_yh_{\perp,x})+0.5p_y(-h_{\perp,y}\partial_yp_z$$
$$+h_{\perp,z}\partial_yp_y+p_y\partial_yh_{\perp,z}-p_z\partial_yh_{\perp,y})-0.5p_z(h_{\perp,y}\partial_zp_z-h_{\perp,z}\partial_zp_y-p_y\partial_zh_{\perp,z}+p_z\partial_zh_{\perp,y})$$
$$+0.5\nu\big[-p_xh_{\perp,x}\partial_yp_z+p_xh_{\perp,x}\partial_zp_y+2p_xh_{\perp,y}\partial_xp_z-2p_xh_{\perp,y}\partial_zp_x-2p_xh_{\perp,z}\partial_xp_y$$
$$+2p_xh_{\perp,z}\partial_yp_x+p_x^2\partial_yh_{\perp,z}-p_x^2\partial_zh_{\perp,y}+p_yh_{\perp,x}\partial_zp_x+p_yh_{\perp,y}\partial_yp_z-2p_yh_{\perp,z}\partial_xp_x-2p_yh_{\perp,z}\partial_yp_y$$

$$
\begin{aligned}
&-p_y h_{\perp,z}\partial_z p_z - 2p_x p_y \partial_x h_{\perp,z} + p_x p_y \partial_z h_{\perp,x} - p_y^2 \partial_y h_{\perp,z} - p_z h_{\perp,x}\partial_y p_x + 2p_z h_{\perp,y}\partial_x p_x \\
&+p_z h_{\perp,y}\partial_y p_y + 2p_z h_{\perp,y}\partial_z p_z - p_z h_{\perp,z}\partial_z p_y + 2p_x p_z \partial_x h_{\perp,y} - p_x p_z \partial_y h_{\perp,x} + p_y p_z \partial_y h_{\perp,y} \\
&-p_y p_z \partial_z h_{\perp,z} + p_z^2 \partial_z h_{\perp,y}\big] + \nu\gamma\lambda\big[ - p_x p_y \partial_y \Delta\mu - p_x p_z \partial_z \Delta\mu + \frac{(-2p_x^2 + p_y^2 + p_z^2)\partial_x \Delta\mu}{3} \\
&+((\frac{-4p_x \partial_x p_x}{3} - p_x \partial_y p_y - p_x \partial_z p_z + \frac{2p_y \partial_x p_y}{3} - p_y \partial_y p_x + \frac{2p_z \partial_x p_z}{3} - p_z \partial_z p_x)\Delta\mu)\big] \\
&+\zeta(\Delta\mu\partial_x q_{xx} + \Delta\mu\partial_y q_{xy} + \Delta\mu\partial_z q_{xz}) + \partial_x \sigma_{xx}^{(e)} + \partial_y \sigma_{xy}^{(e)} + \partial_z \sigma_{xz}^{(e)}\Bigg] + \partial_x \Pi
\end{aligned}
\tag{B.12a}
$$

y-component:

$$
\begin{aligned}
&\eta(2\partial_{yy}v_y + \partial_{xx}v_y + \partial_{xy}v_x + \partial_{yz}v_z + \partial_{zz}v_y) + \nu^2\gamma\Bigg[ \\
&\frac{1}{(p_x^2+p_y^2+p_z^2)^2}\big[ - 2p_x p_y(p_x\partial_x p_x + p_y\partial_x p_y + p_z\partial_x p_z)(p_x^2\partial_x v_x + p_x p_y(\partial_x v_y + \partial_y v_x) + p_y^2\partial_y v_y \\
&\qquad + p_x p_z(\partial_x v_z + \partial_z v_x) + p_y p_z(\partial_y v_z + \partial_z v_y) + p_z^2\partial_z v_z) - 2p_y p_z(p_x\partial_z p_x + p_y\partial_z p_y \\
&\qquad + p_z\partial_z p_z)(p_x^2\partial_x v_x + p_x p_y(\partial_x v_y + \partial_y v_x) + p_y^2\partial_y v_y + p_x p_z(\partial_x v_z + \partial_z v_x) \\
&\qquad + p_y p_z(\partial_y v_z + \partial_z v_y) + p_z^2\partial_z v_z) + \frac{1}{3}(2(p_x\partial_y p_x + p_y\partial_y p_y + p_z\partial_y p_z) \\
&\qquad (p_x^2 - 2p_y^2 + p_z^2)(p_x^2\partial_x v_x + p_x p_y(\partial_x v_y + \partial_y v_x) + p_y^2\partial_y v_y + p_x p_z(\partial_x v_z + \partial_z v_x) \\
&\qquad + p_y p_z(\partial_y v_z + \partial_z v_y) + p_z p_z\partial_z v_z))\big] \\
&+\frac{1}{p_x^2+p_y^2+p_z^2}\big[(p_x\partial_x p_y - \frac{2p_x\partial_y p_x}{3} + p_y\partial_x p_x + \frac{4p_y\partial_y p_y}{3} + p_y\partial_z p_z - \frac{2p_z\partial_y p_z}{3} + p_z\partial_z p_y) \\
&\qquad (p_x^2\partial_x v_x + p_x p_y(\partial_x v_y + \partial_y v_x) + p_y^2\partial_y v_y + p_x p_z(\partial_x v_z\partial_z v_x) + p_y p_z(\partial_y v_z + \partial_z v_y) \\
&\qquad + p_z^2\partial_z v_z) + p_x p_y(2p_x\partial_x p_x\partial_x v_x + p_x^2\partial_{xx}v_x + p_y^2\partial_{xx}v_y + p_z^2\partial_{xx}v_z \\
&\qquad + p_x\partial_x p_y(\partial_x v_y + \partial_y v_x) + p_y\partial_x p_x(\partial_x v_y + \partial_y v_x) + 2p_y\partial_x p_y\partial_y v_y + p_x p_y(\partial_{xx}v_y \\
&\qquad + \partial_{xy}v_x) + p_x\partial_x p_z(\partial_x v_z + \partial_z v_x) + p_z\partial_x p_x(\partial_x v_z + \partial_z v_x) + p_y\partial_x p_z(\partial_y v_z + \partial_z v_y) \\
&\qquad + p_z\partial_x p_y(\partial_y v_z + \partial_z v_y) + 2p_z\partial_x p_z\partial_z v_z + p_x p_z(\partial_{xx}v_z + \partial_{xz}v_x) + p_y p_z(\partial_{xy}v_z \\
&\qquad + \partial_{xz}v_y))) + \frac{1}{3}((-p_x^2 + 2p_y^2 - p_z^2)(2p_x\partial_y p_x\partial_x v_x + p_x^2\partial_{xy}v_x + p_x\partial_y p_y(\partial_x v_y + \partial_y v_x) \\
&\qquad + p_y\partial_y p_x(\partial_x v_y + \partial_y v_x) + 2p_y\partial_y p_y\partial_y v_y + p_x p_y(\partial_{xy}v_y + \partial_{yy}v_x) + p_y^2\partial_{yy}v_y \\
&\qquad + p_z^2\partial_{yz}v_z + p_x\partial_y p_z(\partial_x v_z + \partial_z v_x) + p_z\partial_y p_x(\partial_x v_z + \partial_z v_x) + p_y\partial_y p_z(\partial_y v_z \\
&\qquad + \partial_z v_y) + p_z\partial_y p_y(\partial_y v_z\partial_z v_y) + 2p_z\partial_y p_z\partial_z v_z + p_x p_z(\partial_{xy}v_z + \partial_{zy}v_x) \\
&\qquad + p_y p_z(\partial_{yy}v_z + \partial_{zy}v_y)) + p_y p_z(2p_x\partial_z p_x\partial_x v_x + p_x^2\partial_{xz}v_x + p_x\partial_z p_y(\partial_x v_y + \partial_y v_x) \\
&\qquad + p_y\partial_z p_x(\partial_x v_y + \partial_y v_x) + 2p_y\partial_z p_y\partial_y v_y + p_x p_y(\partial_{xz}v_y + \partial_{yz}v_x) + p_y^2\partial_{yz}v_y
\end{aligned}
$$

$$
\begin{aligned}
&+p_x\partial_z p_z(\partial_x v_z+\partial_z v_x)+p_z\partial_z p_x(\partial_x v_z+\partial_z v_x)+p_y\partial_z p_z(\partial_y v_z+\partial_z v_y)\\
&+p_z\partial_z p_y(\partial_y v_z+\partial_z v_y)+2p_z\partial_z p_z\partial_z v_z+p_x p_z(\partial_{xz}v_z+\partial_{zz}v_x)+p_y p_z(\partial_{yz}v_z\\
&+\partial_{zz}v_y)+p_z^2\partial_{zz}v_z)\Big]\Bigg]\\
=-\Bigg[&-0.5p_x h_{\perp,y}\partial_z p_y-0.5p_x h_{\perp,z}\partial_x p_x-0.5p_x h_{\perp,z}\partial_z p_z+0.5p_y h_{\perp,x}\partial_z p_y-0.5p_y h_{\perp,z}\partial_x p_y\\
&+0.5p_y(h_{\perp,x}\partial_z p_y-h_{\perp,y}\partial_z p_x-p_x\partial_z h_{\perp,y}+p_y\partial_z h_{\perp,x})+0.5p_z h_{\perp,x}\partial_x p_x+0.5p_z h_{\perp,x}\partial_z p_z\\
&+0.5p_z h_{\perp,y}\partial_x p_y+0.5p_x(-h_{\perp,x}\partial_x p_z+h_{\perp,z}\partial_x p_x+p_x\partial_x h_{\perp,z}-p_z\partial_x h_{\perp,x})-0.5p_y(-h_{\perp,y}\partial_x p_z\\
&+h_{\perp,z}\partial_x p_y+p_y\partial_x h_{\perp,z}-p_z\partial_x h_{\perp,y})+0.5p_z(h_{\perp,x}\partial_z p_z-h_{\perp,z}\partial_z p_x-p_x\partial_z h_{\perp,z}+p_z\partial_z h_{\perp,x})\\
&+0.5\nu\big[-p_x h_{\perp,x}\partial_x p_z-p_x h_{\perp,y}\partial_z p_y+2p_x h_{\perp,z}\partial_x p_x+2p_x h_{\perp,z}\partial_y p_y+p_x h_{\perp,z}\partial_z p_z\\
&+p_x^2\partial_x h_{\perp,z}-2p_y h_{\perp,x}\partial_y p_z+2p_y h_{\perp,x}\partial_z p_y+p_y h_{\perp,y}\partial_x p_z-p_y h_{\perp,y}\partial_z p_x-2p_y h_{\perp,z}\partial_x p_y\\
&+2p_y h_{\perp,z}\partial_y p_x+2p_x p_y\partial_y h_{\perp,z}-p_x p_y\partial_z h_{\perp,y}-p_y^2\partial_x h_{\perp,z}+p_y^2\partial_z h_{\perp,x}-p_z h_{\perp,x}\partial_x p_x\\
&+2p_z h_{\perp,x}\partial_y p_y-2p_z h_{\perp,x}\partial_z p_z+p_z h_{\perp,y}\partial_x p_y+p_z h_{\perp,z}\partial_z p_x-p_x p_z\partial_x h_{\perp,x}+p_x p_z\partial_z h_{\perp,z}\\
&+p_y p_z\partial_x h_{\perp,y}-2p_y p_z\partial_y h_{\perp,x}-p_z^2\partial_z h_{\perp,x}\big]+\nu\gamma\lambda\big[-p_x p_y\partial_x\Delta\mu-p_y p_z\partial_z\Delta\mu+\frac{(p_x^2-2p_y^2+p_z^2)\partial_y\Delta\mu}{3}\\
&+(-p_x\partial_x p_y+\frac{2p_x\partial_y p_x}{3}-p_y\partial_x p_x-\frac{4p_y\partial_y p_y}{3}-p_y\partial_z p_z+\frac{2p_z\partial_y p_z}{3}-p_z\partial_z p_y)\Delta\mu)\big]\\
&+\zeta(\Delta\mu\partial_x q_{xy}+\Delta\mu\partial_y q_{yy}+\Delta\mu\partial_z q_{yz})+\partial_x\sigma_{xy}^{(e)}+\partial_y\sigma_{yy}^{(e)}+\partial_z\sigma_{yz}^{(e)}\Bigg]+\partial_y\Pi \qquad \text{(B.12b)}
\end{aligned}
$$

z-component:

$$
\begin{aligned}
&\eta(2\partial_{zz}v_z+\partial_{xx}v_z+\partial_{xz}v_x+\partial_{yy}v_z+\partial_{yz}v_y)+\nu^2\gamma\Bigg[\\
&\frac{1}{(p_x^2+p_y^2+p_z^2)^2}\big[-2p_x p_z(p_x\partial_x p_x+p_y\partial_x p_y+p_z\partial_x p_z)(p_x^2\partial_x v_x+p_x p_y(\partial_x v_y+\partial_y v_x)+p_y^2\partial_y v_y\\
&+p_x p_z(\partial_x v_z+\partial_z v_x)+p_y p_z(\partial_y v_z+\partial_z v_y)+p_z^2\partial_z v_z)-2p_y p_z(p_x\partial_y p_x+p_y\partial_y p_y\\
&+p_z\partial_y p_z)(p_x^2\partial_x v_x+p_x p_y(\partial_x v_y+\partial_y v_x)+p_y^2\partial_y v_y+p_x p_z(\partial_x v_z+\partial_z v_x)\\
&+p_y p_z(\partial_y v_z+\partial_z v_y)+p_z^2\partial_z v_z)+\frac{1}{3}(2(p_x\partial_z p_x+p_y\partial_z p_y+p_z\partial_z p_z)\\
&(p_x^2+p_y^2-2p_z^2)(p_x^2\partial_x v_x+p_x p_y(\partial_x v_y+\partial_y v_x)+p_y^2\partial_y v_y+p_x p_z(\partial_x v_z+\partial_z v_x)\\
&+p_y p_z(\partial_y v_z+\partial_z v_y)+p_z p_z\partial_z v_z))\big]\\
&+\frac{1}{p_x^2+p_y^2+p_z^2}\big[(p_x\partial_x p_z-\frac{2p_x\partial_z p_x}{3}+p_y\partial_y p_z-\frac{2p_y\partial_z p_y}{3}+p_z\partial_x p_x+p_z\partial_y p_y+\frac{4p_z\partial_z p_z}{3})\\
&(p_x^2\partial_x v_x+p_x p_y(\partial_x v_y+\partial_y v_x)+p_y^2\partial_y v_y+p_x p_z(\partial_x v_z\partial_z v_x)+p_y p_z(\partial_y v_z+\partial_z v_y)
\end{aligned}
$$

$$
\begin{aligned}
&+ p_z^2\partial_z v_z) + p_x p_z(2p_x\partial_x p_x\partial_x v_x + p_x^2\partial_{xx}v_x + p_y^2\partial_{xx}v_y + p_z^2\partial_{xx}v_z \\
&+ p_x\partial_x p_y(\partial_x v_y + \partial_y v_x) + p_y\partial_x p_x(\partial_x v_y + \partial_y v_x) + 2p_y\partial_x p_y\partial_y v_y + p_x p_y(\partial_{xx}v_y \\
&+ \partial_{xy}v_x) + p_x\partial_x p_z(\partial_x v_z + \partial_z v_x) + p_z\partial_x p_x(\partial_x v_z + \partial_z v_x) + p_y\partial_x p_z(\partial_y v_z + \partial_z v_y) \\
&+ p_z\partial_x p_y(\partial_y v_z + \partial_z v_y) + 2p_z\partial_x p_z\partial_z v_z + p_x p_z(\partial_{xx}v_z + \partial_{xz}v_x) + p_y p_z(\partial_{xy}v_z \\
&+ \partial_{xz}v_y))) + p_y p_z(2p_x\partial_y p_x\partial_x v_x + p_x^2\partial_{xy}v_x + p_x\partial_y p_y(\partial_x v_y + \partial_y v_x) \\
&+ p_y\partial_y p_x(\partial_x v_y + \partial_y v_x) + 2p_y\partial_y p_y\partial_y v_y + p_x p_y(\partial_{xy}v_y + \partial_{yy}v_x) + p_y^2\partial_{yy}v_y \\
&+ p_z^2\partial_{yz}v_z + p_x\partial_y p_z(\partial_x v_z + \partial_z v_x) + p_z\partial_y p_x(\partial_x v_z + \partial_z v_x) + p_y\partial_y p_z(\partial_y v_z \\
&+ \partial_z v_y) + p_z\partial_y p_y(\partial_y v_z\partial_z v_y) + 2p_z\partial_y p_z\partial_z v_z + p_x p_z(\partial_{xy}v_z + \partial_{zy}v_x) \\
&+ p_y p_z(\partial_{yy}v_z + \partial_{zy}v_y)) + \frac{1}{3}((-p_x^2 - p_y^2 + 2p_z^2)(2p_x\partial_z p_x\partial_x v_x + p_x^2\partial_{xz}v_x + p_x\partial_z p_y(\partial_x v_y \\
&+ \partial_y v_x) + p_y\partial_z p_x(\partial_x v_y + \partial_y v_x) + 2p_y\partial_z p_y\partial_y v_y + p_x p_y(\partial_{xz}v_y + \partial_{yz}v_x) + p_y^2\partial_{yz}v_y \\
&+ p_x\partial_z p_z(\partial_x v_z + \partial_z v_x) + p_z\partial_z p_x(\partial_x v_z + \partial_z v_x) + p_y\partial_z p_z(\partial_y v_z + \partial_z v_y) \\
&+ p_z\partial_z p_y(\partial_y v_z + \partial_z v_y) + 2p_z\partial_z p_z\partial_z v_z + p_x p_z(\partial_{xz}v_z + \partial_{zz}v_x) + p_y p_z(\partial_{yz}v_z \\
&+ \partial_{zz}v_y) + p_z^2\partial_{zz}v_z)\Big]\Bigg]
\end{aligned}
$$

$$
\begin{aligned}
= -\Bigg[&0.5p_x h_{\perp,y}\partial_x p_x + 0.5p_x h_{\perp,y}\partial_y p_y + 0.5p_x h_{\perp,z}\partial_y p_z - 0.5p_y h_{\perp,x}\partial_x p_x - 0.5p_y h_{\perp,x}\partial_y p_y \\
&-0.5p_y h_{\perp,z}\partial_x p_z + 0.5p_x(-h_{\perp,x}\partial_x p_y + h_{\perp,y}\partial_x p_x + p_x\partial_x h_{\perp,y} - p_y\partial_x h_{\perp,x}) - 0.5p_y(h_{\perp,x}\partial_y p_y \\
&-h_{\perp,y}\partial_y p_x - p_x\partial_y h_{\perp,y} + p_y\partial_y h_{\perp,x}) - 0.5p_z h_{\perp,x}\partial_y p_z + 0.5p_z h_{\perp,y}\partial_x p_z + 0.5p_z(h_{\perp,y}\partial_x p_z \\
&-h_{\perp,z}\partial_x p_y - p_y\partial_x h_{\perp,z} + p_z\partial_x h_{\perp,y}) + 0.5p_z(h_{\perp,x}\partial_y p_z - h_{\perp,z}\partial_y p_x - p_x\partial_y h_{\perp,z} + p_z\partial_y h_{\perp,x}) \\
&+0.5\nu\big[p_x h_{\perp,x}\partial_x p_y - 2p_x h_{\perp,y}\partial_x p_x - p_x h_{\perp,y}\partial_y p_y - 2p_x h_{\perp,y}\partial_z p_z + p_x h_{\perp,z}\partial_y p_z \\
&-p_x^2\partial_x h_{\perp,y} + p_y h_{\perp,x}\partial_x p_x + 2p_y h_{\perp,x}\partial_y p_y + 2p_y h_{\perp,x}\partial_z p_z - p_y h_{\perp,y}\partial_y p_x - p_y h_{\perp,z}\partial_x p_z \\
&+p_x p_y\partial_x h_{\perp,x} - p_x p_y\partial_y h_{\perp,y} + p_y^2\partial_y h_{\perp,x} - 2p_z h_{\perp,x}\partial_y p_z + 2p_z h_{\perp,x}\partial_z p_y + 2p_z h_{\perp,y}\partial_x p_z \\
&-2p_z h_{\perp,y}\partial_z p_x - p_z h_{\perp,z}\partial_x p_y + p_z h_{\perp,z}\partial_y p_x + p_x p_z\partial_y h_{\perp,z} - 2p_x p_z\partial_z h_{\perp,y} - p_y p_z\partial_x h_{\perp,z} \\
&+2p_y p_z\partial_z h_{\perp,x} - p_z^2\partial_x h_{\perp,y} - p_z^2\partial_y h_{\perp,x}\big] + \nu\gamma\lambda\big[-p_x p_z\partial_x\Delta\mu - p_y p_z\partial_y\Delta\mu + \frac{(p_x^2 + p_y^2 - 2p_z^2)\partial_z\Delta\mu}{3} \\
&+(-p_x\partial_x p_z + \frac{2p_x\partial_z p_x}{3} - p_y\partial_y p_z + \frac{2p_y\partial_z p_y}{3} - p_z\partial_x p_x - p_z\partial_y p_y - \frac{4p_z\partial_z p_z}{3})\Delta\mu)\big] \\
&+\zeta(\Delta\mu\partial_x q_{xz} + \Delta\mu\partial_y q_{yz} + \Delta\mu\partial_z q_{zz}) + \partial_x\sigma_{xz}^{(e)} + \partial_y\sigma_{yz}^{(e)} + \partial_z\sigma_{zz}^{(e)}\Bigg] + \partial_z\Pi \qquad \text{(B.12c)}
\end{aligned}
$$

# Appendix C

# Numerical details of 3D actin flow simulations in a spherical shell

We describe the numerical details for solving the actin-flow model equations (C.1a-b) in a spherical shell.

## C.1 One-dimensional solver in the spherically symmetric case

To simplify the discussion, we now consider the spherically-symmetric case, for which the actin density $\rho = \rho(r)$ is a function of the radial distance to the centre $r = |\mathbf{r}|$, and the actin velocity $\mathbf{v} = v(r)\hat{\mathbf{r}}$ is purely radial with $\hat{\mathbf{r}} = \mathbf{r}/r$.

In this case, actin continuity equation (5.1) and force balance (5.2) can be rewritten as:

$$\partial_t \rho + \frac{2v\rho}{r} + \partial_r(\rho v) = f(\rho)\,, \tag{C.1a}$$

$$\partial_r \left[ \frac{2v}{r}(\bar{\eta} - 2\eta/3) + (4\eta/3 + \bar{\eta})\partial_r v \right] + \frac{4\eta}{r}(\partial_r v - v/r) - \partial_r P = \gamma v\,. \tag{C.1b}$$

In the spherically-symmetric case, the differential equations are effectively one-dimensional and we solve them numerically using finite-difference schemes. We implemented both a steady-state and a dynamical solver.

### C.1.1 Steady-state solver

The steady-state solver is implemented in the following way:

- The one-dimensional (dimensionless) interval $[\frac{R_1}{R_2}, 1]$ is discretized using $N$ points: $r \to \{r_1 = \frac{R_1}{R_2}, \cdots, r_N = 1\}$. We typically use $N = 100$, although larger values of $N$ were used to check convergence,

- The velocity and density profiles are discretized on this interval: $v(r) \to \{v_1, \cdots, v_N\}$ and $\rho(r) \to \{\rho_1, \cdots, \rho_N\}$,
- Using finite differences to express the spatial derivative $\partial_r$, Eq. (C.1b) and the steady-state version of Eq. (C.1a) are discretized on the interval and transformed into a nonlinear system of $2N$ equations over the variables $v_i$ and $\rho_i$,
- Velocity boundary conditions (5.10) and the density boundary condition (5.8) are imposed. (Using stress free-boundary conditions (5.12) provides equally good solutions.)
- The nonlinear set of equations is solved using the *FindRoot* solver of *Mathematica* [213]. The initial guess for the solver is a linear density profile $\rho(r) = a + br$ and a velocity profile given by solving analytically the differential equation (C.1b) for this linear density profile, for constant viscosities and friction coefficient, and for $P = a' + b'\rho$.

### C.1.2 Dynamical solver

The dynamical solver uses a finite-difference discretization for the spatial derivatives and an upwind scheme for time stepping. It is implemented in *Python* as follows:

- The one-dimensional (dimensionless) interval $[\frac{R_1}{R_2}, 1]$ is discretized using $N$ points: $r \to \{r_1 = \frac{R_1}{R_2}, \cdots, r_N = 1\}$. We typically use $N = 100$, although larger values of $N$ were used to check convergence,
- Time is discretized: $t \to t_n = n\mathrm{d}t$. We typically use $\mathrm{d}t = 10^{-3}$, although smaller values were also used to check convergence,
- The velocity and density profiles are discretized on this interval: $v(r,t) \to \{v_1(t_n), \cdots, v_N(t_n)\}$ and $\rho(r,t) \to \{\rho_1(t_n), \cdots, \rho_N(t_n)\}$,
- Starting from a linear density profile $\rho(r, t = 0) = a + br$ as initial condition, the time stepping is performed in the following manner:
  1. Using finite differences to express the spatial derivative $\partial_r$, the force balance equation (C.1b) is discretized and forms a linear system of $N$ equations for the velocity $v_i(t_n)$, which are solved using the boundary conditions (5.10). (Stress-free boundary conditions (5.12) were also used and provided equally good results),
  2. The density profile is time-stepped using an upwind scheme and a discretized version of Eq. (C.1a). The density boundary condition $\rho(r_N, t_n) = \rho_2$ is imposed[1],
  3. time is updated to $t_{n+1}$ and the time stepping procedures continues from step 1,

[1] Using the density boundary condition at $r = R_1$, Eq. (5.8) is unstable, since the velocity is directed inwards in our system.

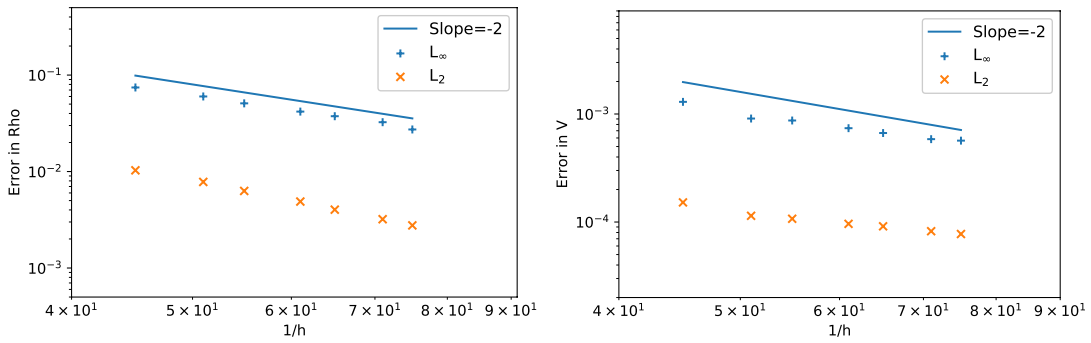



Figure C.1: Convergence of 3D dynamical solver to high-resolution numerical solution obtained form 1D steady state solver. **(a)** Convergence of the actin density. **(b)** Convergence of the velocity field.

- The simulation ends when the density profile has converged up to a prescribed tolerance.

We use the dynamical solver to check that the steady-state solutions found by the steady-state solver are both stable and accessible from reasonable initial conditions.

## C.2 Three-dimensional solver

The three-dimensional dynamical solver is implemented in the scalable computing library OpenFPM [64, 102] using the Discretization-Corrected Particle Strength Exchange method (DC-PSE) [77]:

- The 3D domain is discretized using Eulerian particles with uniform spacing for the domain between $(r_1, r_2)$. The spherical surface at $r_1$ is discretized by placing particles uniformly along the polar coordinates $(r_1, \theta, \phi)$, and at $r_2$ using the Fibonacci distribution technique. Approximately $N = 55$ was used which correspond to an average inter-particle spacing of 0.126296, although smaller spacing were used to check convergence.
- Time is discretized: $t \to t_n$ using the adaptive Adams-Bashforth-Moulten explicit predictor-corrector time stepping method with a tolerance of $10^{-5}$.
- Starting from an initial density profile, time stepping was performed in the following manner:
    1. using DC-PSE for the spatial derivatives, the force balance equation (5.2) is discretized and forms a linear system of equations for the velocity $v_i(t_n)$, which are solved using the boundary conditions (5.10).

2. The density profile is time-stepped using Eq. (5.1). The density boundary condition $\rho(r_N, t_n) = \rho_2$ is imposed.
3. Time is updated and time-stepping is continued.

- The simulation ends when the density profile has converged up to a prescribed tolerance.

The 3D dynamical solver is validated against the one-dimensional steady state solver as shown in Fig. C.1.