

# OpenCMP: An Open-Source Computational Multiphysics Package

by

Elizabeth Julia Monte

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Master of Applied Science  
in  
Chemical Engineering

Waterloo, Ontario, Canada, 2021

© Elizabeth Julia Monte 2021

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Statement of Contributions

All work in this thesis was performed under the guidance of Dr. Nasser Abukhdeir, who suggested the project, helped define its scope, and coined the name `OpenCMP`.

Alex Vasile and James Lowman have both contributed to the `OpenCMP` code base. Alex Vasile wrote the `MultiComponentINS` model and parallelized the `post_processing` module. He wrote several of the unit tests, particularly those for `MultiComponentINS` and those for the helper modules. Alex made many suggestions to optimize the code structure and wrote various in-code documentation. Finally, Alex contributed to the derivations of the discontinuous Galerkin formulations of `OpenCMP` models given in Appendix A.

James Lowman wrote the original code for converting `.sol` files into `.pvd`. He also clarified the proper implementation of normal stress boundary conditions for `INS`-based models. James contributed to the derivations of the discontinuous Galerkin formulations of `OpenCMP` models given in Appendix A. He also contributed to the development of the diffuse interface formulations given in Appendix B.

Finally, Alex Vasile, James Lowman, Chahat Aggarwal, and Ittisak Promma have all been early users of `OpenCMP` and have given valuable feedback for its improvement.

## Abstract

Computational multiphysics offers a safe, inexpensive, and rapid alternative to direct experimentation, but there remain barriers to its widespread use.

One such barrier is the generation of conformal meshes of simulation domains, which is the primary approach to spatial discretization used in multiphysics simulations. Generation of these meshes is time intensive, non-deterministic, and often requires manual user intervention. For complex domain geometries, there is also a competition between domain-conforming mesh elements, element and mesh quality within the domain, and simulation stability.

A second barrier is lack of easy access to computational multiphysics software based on the finite element method, which enables high-order spatial discretization at the cost of complexity of implementation. Most computational multiphysics software is based on the finite volume method, which involves inherently low-order spatial discretization. This requires relatively high densities of mesh elements for adequate numerical accuracy but is relatively simple to implement. However, higher mesh densities correspond to smaller element scales, resulting in stability issues for convection-dominated simulations. There do exist finite element method-based computational multiphysics software packages, however these software packages are either closed-source or require extensive user skills in a broad range of areas including continuum mechanics, applied math, and computational science.

This thesis presents **OpenCMP**, a new open-source computational multiphysics package. **OpenCMP** implements the diffuse interface method, which allows even complex geometries to be meshed with nonconforming structured grids, improving simulation stability and sometimes speed. **OpenCMP** is built on the popular finite element library, **NGSolve**, and offers both a simple user interface for running standard models and the ability for experienced users to easily add new models. It has been validated on common benchmark problems and used to extend the diffuse interface method to simulations with moving domains.



## Acknowledgements

Firstly, I would like to thank my supervisor Dr. Nasser Abukhdeir for his support and guidance throughout all of my work with him. I would also like to thank all the members of the COMPHYS group for their friendship, advice, and moral support.

Many thanks to Alex Vasile and James Lowman for their contributions to `OpenCMP`. The `OpenCMP` code base would be a far messier product without Alex's expertise and model development would have been a far longer struggle without James and Alex to bounce ideas off of.

Finally, I would like to acknowledge the following sources of funding: the University of Waterloo Engineering Excellence Master's Fellowship and the Natural Sciences and Engineering Research Council of Canada Canada Graduate Scholarship Master's Program. Further thanks to Compute Canada for access to their high performance computing resources.

# Table of Contents

List of Tables	x
List of Figures	xi
<b>1 Introduction</b>	<b>1</b>
1.1 Research Motivation . . . . .	1
1.2 Objectives . . . . .	2
1.3 Structure of Thesis . . . . .	3
<b>2 Background</b>	<b>5</b>
2.1 Computational Multiphysics in Chemical Engineering . . . . .	5
2.2 Common Numerical Methods . . . . .	6
2.2.1 The Finite Element Method . . . . .	7
2.2.2 The Discontinuous Galerkin Method . . . . .	9
2.3 The Diffuse Interface Method . . . . .	13
<b>3 Related Work</b>	<b>17</b>
3.1 Design Considerations . . . . .	17
3.1.1 Models and Numerical Methods . . . . .	17
3.1.2 Location in the Simulation Workflow . . . . .	18
3.1.3 User Interface . . . . .	20

3.1.4	Closed- or Open-Source	25
3.2	Current Options	26
3.2.1	Commercial Software Packages	27
3.2.2	Commercial Platforms and Toolboxes	27
3.2.3	Open-Source Software Packages	28
3.2.4	Open-Source Solver Libraries	29
3.2.5	Auxiliary Software	30
<b>4</b>	<b>Overview of OpenCMP</b>	<b>32</b>
4.1	Solver Back-end	32
4.2	Code Structure	33
4.2.1	Basic Workflow	33
4.2.2	<code>run</code>	34
4.2.3	<code>Solver</code>	34
4.2.4	<code>Model</code>	35
4.2.5	<code>ConfigFunctions</code>	36
4.2.6	<code>DIM</code>	36
4.2.7	<code>ConfigParser</code>	37
4.2.8	<code>error_analysis</code> and <code>error</code>	37
4.2.9	<code>post_processing</code>	37
4.3	User Interface	37
4.3.1	Configuration Files	38
4.3.2	Command Line Interaction	42
4.4	Capabilities	42
<b>5</b>	<b>Performance Verification</b>	<b>45</b>
5.1	The Poisson Equation	47
5.1.1	The Steady-State Poisson Equation	47

5.1.2	The Transient Poisson Equation . . . . .	48
5.2	The Stokes Equations . . . . .	50
5.2.1	Poiseuille Flow . . . . .	50
5.2.2	Couette Flow . . . . .	52
5.2.3	The Transient Stokes Equations . . . . .	54
5.3	The Incompressible Navier-Stokes Equations . . . . .	58
5.3.1	Taylor-Green Vortices . . . . .	58
5.3.2	Schäfer-Turek Benchmark . . . . .	62
5.4	Multi-Component Flow . . . . .	66
5.4.1	Multi-Component Diffusion . . . . .	66
5.4.2	Multi-Component Convection and Diffusion . . . . .	67
5.4.3	Reacting Flow . . . . .	69
5.5	Time Discretization Schemes . . . . .	72
5.5.1	Fixed Time Step Schemes . . . . .	72
5.5.2	Adaptive Time-Stepping . . . . .	74
5.6	Comparison to Existing Software . . . . .	75
<b>6</b>	<b>Extension of the Diffuse Interface Method</b>	<b>79</b>
6.1	Use of Impellers in Chemical Engineering . . . . .	79
6.2	Phase Field Motion . . . . .	81
6.3	Diffuse Interface Simulation of a Stirred Tank Reactor . . . . .	84
<b>7</b>	<b>Conclusions</b>	<b>89</b>
7.1	Conclusions . . . . .	89
7.2	Recommendations . . . . .	90
	<b>References</b>	<b>92</b>
	<b>APPENDICES</b>	<b>101</b>

<b>A</b>	<b>Discontinuous Galerkin Formulations</b>	<b>102</b>
A.1	The Poisson Equation . . . . .	102
A.2	The Stokes Equations . . . . .	108
A.3	The Incompressible Navier-Stokes Equations . . . . .	116
A.3.1	IMEX Time Discretization . . . . .	126
<b>B</b>	<b>Diffuse Interface Formulations</b>	<b>131</b>
B.1	The Poisson Equation . . . . .	131
B.2	The Stokes Equations . . . . .	134
B.3	The Incompressible Navier-Stokes Equations . . . . .	137

# List of Tables

4.1	Current capabilities of <code>OpenCMP</code> . . . . .	44
5.1	Error results for Poiseuille flow. . . . .	51
5.2	Error results for Couette flow. . . . .	53
5.3	Error results for multi-component diffusion. . . . .	67
5.4	Comparison to <code>OpenFOAM</code> <sup>®</sup> . . . . .	77
6.1	Geometry of the stirred tank reactor and Rushton impeller. . . . .	84

# List of Figures

2.1	Example finite element method piecewise continuous trial and weighting functions. In one dimension each mesh element $\Omega_k$ has two nodes and thus two interpolating polynomials. . . . .	9
2.2	Example comparison of the trial functions of the finite element method (FEM) and the discontinuous Galerkin method (DG). . . . .	10
2.3	Adaptive mesh refinement with (a) hanging nodes (highlighted in blue) and (b) non-hanging nodes. The use of non-hanging nodes requires the addition of triangular mesh elements to the structured quadrilateral mesh. . . . .	12
2.4	An example simulation domain (blue) meshed by increasingly refined unstructured conformal meshes. Conformance of mesh elements to the simulation domain boundary improves with increasing refinement. . . . .	13
2.5	Examples of (a) an unstructured triangular mesh, (b) a structured triangular mesh, and (c) a structured quadrilateral mesh. . . . .	13
2.6	Example of (a) a complex geometry meshed by a non-conforming mesh (mesh coarsened for visibility), (b) the phase field approximation of the geometry, and (c) the phase field approximation of the complex geometry boundary. Also the approximation of (e) the Heaviside function by the phase field and (f) the Dirac delta function by the magnitude of the gradient of the phase field. . . . .	16
3.1	An example <code>NGSolve</code> script to solve the Poisson equation. . . . .	21
3.2	Portion of an example <code>SU2</code> configuration file for flow past an airfoil. . . . .	22
3.3	An example of the <code>COMSOL Multiphysics®</code> graphical user interface. . . . .	24
4.1	The code structure of <code>OpenCMP</code> . Solid lines indicate imported modules and dashed lines indicate initialized classes. . . . .	33

4.2	The inheritance structure of the <code>Solver</code> class. . . . .	35
4.3	The inheritance structure of the <code>Model</code> class. . . . .	36
4.4	The inheritance structure of the <code>ConfigFunctions</code> class. . . . .	36
4.5	An example simulation directory structure. . . . .	39
4.6	An example configuration file for the main simulation directory. . . . .	40
4.7	An example configuration file for the boundary condition sub-directory. . . . .	41
4.8	Command line output of an example <code>OpenCMP</code> simulation. . . . .	42
5.1	Mesh convergence for the Poisson equation. . . . .	47
5.2	Simulation results for the Poisson equation. . . . .	48
5.3	(a) mesh convergence and (b) time step convergence for the transient Poisson equation. . . . .	49
5.4	Simulation results at (a) $t = 0.1$ s, (b) $t = 0.5$ s, and (c) $t = 0.9$ s for the transient Poisson equation. . . . .	50
5.5	Simulation results of (a) velocity and (b) pressure for Poiseuille flow. . . . .	52
5.6	Simulation results of (a) velocity and (b) pressure for Couette flow. . . . .	54
5.7	Mesh convergence for (a) velocity and (b) pressure and time step convergence for (c) velocity and (d) pressure for transient Stokes flow. . . . .	56
5.8	Simulation results for velocity at (a) $t = 0$ s, (b) $t = 0.5$ s, and (c) $t = 1$ s for transient Stokes flow. . . . .	57
5.9	Simulation results for pressure at (a) $t = 0$ s, (b) $t = 0.5$ s, and (c) $t = 1$ s for transient Stokes flow. . . . .	57
5.10	Mesh convergence for (a) velocity and (b) pressure and time step convergence for (c) velocity and (d) pressure for Taylor-Green vortices. . . . .	60
5.11	Simulation results for velocity at (a) $t = 0$ s, (b) $t = 0.5$ s, and (c) $t = 1$ s for Taylor-Green vortices. . . . .	61
5.12	Simulation results for pressure at (a) $t = 0$ s, (b) $t = 0.5$ s, and (c) $t = 1$ s for Taylor-Green vortices. . . . .	61
5.13	The simulation geometry for the two-dimensional Schäfer-Turek benchmark. . . . .	62
5.14	Plots of the (a) drag coefficient and (b) lift coefficient from initialization to steady vortex-shedding. . . . .	63



5.15	Simulation results of (a) velocity and (b) pressure for the two dimensional Schäfer-Turek benchmark. . . . .	64
5.16	The simulation geometry for one of the three-dimensional Schäfer-Turek benchmarks. . . . .	65
5.17	Simulation result of the concentrations of components A, B, and C for multi-component diffusion. . . . .	67
5.18	Mesh convergence for multi-component convection and diffusion. . . . .	68
5.19	Simulation result of the concentration of component A for multi-component convection and diffusion. . . . .	69
5.20	Simulation results of the concentrations of (a) component A and (b) component B for reacting flow. . . . .	71
5.21	Simulation results of (a) velocity and (b) pressure for reacting flow. . . . .	72
5.22	Time step convergence for (a) velocity and (b) pressure for the various available fixed time step time discretization schemes. . . . .	73
5.23	Plots of (a) the maximum inlet velocity and pressure and (b) the time step used by the adaptive time-stepping scheme. . . . .	75
5.24	Simulation results for velocity from (a) <code>OpenFOAM</code> <sup>®</sup> , (b) <code>OpenCMP</code> using the standard finite element method, and (c) <code>OpenCMP</code> using the discontinuous Galerkin method for the lid-driven cavity. . . . .	77
5.25	Comparison to the reference solutions of (a) the velocity x-component along $y = 0.05$ and (b) the velocity y-component along $x = 0.05$ . . . . .	78
6.1	Diagram of a basic stirred tank batch reactor. . . . .	80
6.2	Phase field location at (a) $t = 0$ s, (b) $t = 0.4$ s, and (c) $t = 0.9$ s. The phase field is one in the fluid domain and zero in the solid domain. . . . .	82
6.3	Simulation results of (a) velocity and (b) pressure for two-dimensional rigid body rotation with the diffuse interface method. . . . .	83
6.4	(a) top and (b) side view of the Rushton impeller and (c) tank geometry. . . . .	85
6.5	Simulation results for velocity in the (a) xy-plane and (b) xz-plane for a stirred tank reactor. . . . .	87
6.6	Simulation results for pressure in the (a) xy-plane and (b) xz-plane for a stirred tank reactor. . . . .	88

# Chapter 1

## Introduction

### 1.1 Research Motivation

Simulations have the potential to play a key role in the engineering design process. They offer an inexpensive, rapid, and safe alternative to physical prototyping for preliminary design screening and optimization. Simulations also offer detailed insight into the physical phenomena of interest, often beyond the capabilities of feasible experimentation. As such, there is significant interest—in academia, government, and industry—in increasing the usage of simulations, the computing infrastructure available to run them, and advancing their predictive capabilities [1]. Computational multiphysics is of particular interest as almost all engineering applications involve coupled physical and chemical phenomena [2].

However, computational multiphysics simulations are currently infeasible for high-throughput design screening of most industrially-relevant processes. Typical processes involve complex domain geometries that are time and labour intensive to conformally mesh [3, 4]. The resulting meshes are often unstructured which increases computational complexity compared to similar sized structured meshes [5] and may introduce numerical inaccuracies and instabilities into the simulations [4].

Immersed boundary methods allow the use of simple structured meshes for any geometry, with the geometry boundary located by a marker field instead of the boundary of the mesh itself [6]. This offers significant stability benefits, particularly for convective-dominated processes. However, previous work on immersed boundary methods has focused on obtaining equivalent accuracy to conformal mesh simulations, resulting in immersed boundary simulations which are just as computationally intensive, if not more so (for example, see work by Stoter *et al* [7]). Yet, such accuracy is unnecessary for preliminary

design screening; simulations need only be sufficiently accurate to distinguish between different candidate designs in order to select relatively optimal designs for further analysis. This is the focus of the diffuse interface method as implemented by Monte *et al* [8].

The diffuse interface method is a phase field-based immersed boundary method, where a phase field is a continuous field which quantifies a physical characteristic of a material which may be used to infer its phase [9]. Said implementation provides fast automated mesh generation and has been shown to give simulation results which are sufficiently accurate for design screening in an order-of-magnitude less time than standard conformal mesh simulations [8]. It is well suited to screening large numbers of initial candidate designs and coarse-grained design optimization.

A publicly available implementation of the diffuse interface method is desirable to enable replication and further work. Incorporating it into a computational multiphysics software package would make it easily accessible to the general simulation community. There is also a need for easily accessible computational multiphysics software based on the finite element method. Historically, computational multiphysics software has focused on the finite volume method for fluid dynamics simulations, mainly due to its property of local conservation and stability. However, the finite element method offers many advantages over the finite volume method such as the easy use of high-order polynomial interpolants to improve simulation accuracy [10]. The finite element method can even be made locally conservative through the use of the discontinuous Galerkin method [11]. The commercial finite element-based computational multiphysics software packages that exist are closed-source [12]. Thus, their numerical implementations are hidden, which hampers replication of research, prevents users from modifying or adding new models, and hides possibly undesirable (with respect to accuracy and/or correctness) implementation details. Existing open-source finite element-based computational multiphysics software has a steep learning curve and is inaccessible to the general simulation community [13, 14, 15, 16, 17, 18, 19].

In summary, to enable the research community and general simulation community, a computational multiphysics software package is needed which is user friendly, based on the finite element method, and provides a publicly available implementation of the diffuse interface method.

## 1.2 Objectives

The objective of this work is to develop the software package described in the previous section. Said package, called `OpenCMP`, should fulfill the following criteria:

**Functionality** The diffuse interface method should be incorporated. It and a discontinuous Galerkin formulation should be available for every model.

**Usability** `OpenCMP` should be very user friendly and typical use should require minimal coding experience. It should support the main simulation workflow including basic post-processing.

**Extensibility** `OpenCMP` should have a highly abstracted code structure to allow users to easily implement their own models or modify existing models. It should interface with widely used third-party packages for mesh generation and for more extensive visualization and analysis of simulation results.

Standard benchmark problems will be used to verify the performance of `OpenCMP`.

`OpenCMP` will also be used to extend work on the diffuse interface method as follows:

1. Extension of the diffuse interface method to simulations with moving domains through the use of a moving phase field.
2. Use of the diffuse interface method to simulate the rigid body rotation of an impeller in a stirred tank reactor.
3. Comparison of the steady-state flow distribution to published experimental results.

## 1.3 Structure of Thesis

This work is organized into seven chapters: Chapter 2—background, Chapter 3—related work, Chapter 4—overview of `OpenCMP`, Chapter 5—performance verification, Chapter 6—extension of the diffuse interface method, and Chapter 7—conclusions.

Chapter 2 discusses the role of computational multiphysics in chemical engineering. It then reviews the numerical methods used in this work, particularly the finite element method and the discontinuous Galerkin method. The diffuse interface method is also described.

Chapter 3 discusses key design considerations for a computational multiphysics software package and reviews popular existing packages.

Chapter 4 gives an overview of `OpenCMP`. The finite element solver back-end and the main components of the code structure are described. Then examples are given of the user interface and user interaction through the command line. Finally, the simulation capabilities of `OpenCMP` are discussed.

Chapter 5 details performance verification of the numerical and code implementation of `OpenCMP`. All models and time discretization schemes are assessed for accuracy and error

convergence rates. A timing and accuracy comparison to the popular computational fluid dynamics software package `OpenFOAM`<sup>®</sup> [20] is also given.

Chapter 6 describes the extension of the diffuse interface method to simulations with moving domains. The diffuse interface is used to model impeller rotation in the simulation of a stirred tank reactor without mesh movement or re-meshing.

Chapter 7 summarizes conclusions from this work and recommends future additions to `OpenCMP` and the diffuse interface method.

# Chapter 2

## Background

This chapter provides a brief background on computational multiphysics and the numerical methods used in OpenCMP.

### 2.1 Computational Multiphysics in Chemical Engineering

Computational multiphysics is the use of numerical simulations to model complex coupled processes. This is most commonly taken to mean systems governed by multiple different physical or chemical phenomena. It can also refer to multiscale systems, where different processes occur on significantly different length scales or time scales [21]. Some common multiphysics processes include thermomechanics—coupling of heat transfer with structural mechanics—hydromechanics—coupling of fluid and structural mechanics—electromagnetics—coupling of electrostatics and magnetostatics—and elastodynamics—dynamics of elastic bodies [2].

Most computational multiphysics problems in chemical engineering revolve around coupled fluid dynamics, heat and mass transfer, and chemical reaction. Common applications are chemical reactors and bioreactors, see for example work by Jimeno *et al* [22] and by Tamrakar and Ramachandran [23], where chemical reaction, at catalysts or within biological organisms, occurs in the presence of fluid flow and heat transfer. Reaction efficiency is often highly dependent on optimal flow rates, mixing, and operating temperatures. Photon transport may also need to be considered in the case of photocatalysts as in work by Casado *et al* [24], or extensive biokinetics and growth models as in work by Rajabzadeh *et*

*al* [25]. Wastewater treatment involves similar physical and chemical phenomena, see for example work by Ghadiri *et al* [26], and may further include models and closures for flow through porous media.

All of these processes, and many more in chemical engineering, are governed by the laws of physics and thermodynamics in the form of the conservation of mass, momentum, and energy. Furthermore, most chemical engineering processes operate at length scales far larger than molecular length scales. Thus, they are adequately represented by continuum models such as the Navier-Stokes equations and the equation of change of temperature (for Newtonian fluids), with additions and modifications for multi-component or multi-phase flows, turbulence, and reaction [27].

## 2.2 Common Numerical Methods

The continuum models mentioned in the previous section form systems of partial differential equations. There are very few cases where these equations can be solved analytically, instead their solutions must be approximated numerically.

The most common numerical methods in computational multiphysics are the finite difference method, the finite volume method, and the finite element method [28]. All three methods involve the discretization of the simulation domain into many smaller domains by a mesh then the use of local interpolants—defined on mesh nodes or mesh elements—to represent the solution over the global domain. The finite difference method uses a Taylor series expansion to locally approximate the derivatives of the differential equation in terms of solution values at nearby mesh nodes. It is very simple to implement and very effective on Cartesian grids. However, the finite difference method is very difficult to extend to the complex domains typical of real-world applications. The finite volume method uses a zeroth-order approximation of the solution within each mesh element and allows this approximation to vary discontinuously between mesh elements, rendering the method very stable. The specific value of the approximate solution within each mesh element is determined from flux balances over each mesh element and the continuity of fluxes between neighbouring mesh elements. This also renders the finite volume method locally conservative. In contrast, the finite element method uses higher-order polynomial interpolants within mesh elements and enforces continuity of the approximate solution across mesh element nodes without constraining fluxes. This improves accuracy relative to the finite volume method at the cost of stability and local conservation [28].

Historically, the finite volume method has seen the most use for applications involving fluid flow, while the finite element method has primarily been used for structural mechanics

simulations [28]. However, as will be discussed, recent advances to the finite element method resolve its main deficiencies while retaining its attractive qualities relative to the other methods.

### 2.2.1 The Finite Element Method

The finite element method can be derived from the method of weighted residuals following the process given by Rice and Do [29]. Consider the Poisson equation:

$$\nabla^2 u + f = 0 \tag{2.1}$$

defined on a domain  $\Omega$  with appropriate boundary conditions. An approximate solution (or trial function) can be constructed from a linear combination of interpolating polynomials:

$$u_A(\mathbf{x}) = u_0(\mathbf{x}) + \sum_{i=1}^N a_i \psi_i(\mathbf{x}) \tag{2.2}$$

where  $u_0(\mathbf{x})$  must be chosen to satisfy the boundary conditions. Substituting the approximate solution into the original differential equation gives a residual error:

$$\nabla^2 u_A + f = \sum_{i=1}^N a_i \nabla^2 \psi_i + f = R(\mathbf{x}) \tag{2.3}$$

which will only be zero when the approximate solution is the exact solution to the differential equation. Thus, the trial function coefficients  $a_i$  should be chosen to minimize the residual and give the best approximation to the exact solution.

As shown in eqn. (2.3), the residual can vary spatially so it is challenging to minimize it over the entire domain simultaneously. Instead, the residual is averaged over the domain and the resulting scalar value is minimized. From here on, a spatially-varying residual will be identified by  $R(\mathbf{x})$  while the scalar value obtained from averaging is simply  $R$ . A set of weighting functions  $\{v_k(\mathbf{x})\}$  (or test functions) can be included to control where in the domain the residual is minimized and improve the accuracy of the final approximate solution:

$$\int_{\Omega} R(\mathbf{x}) v_k(\mathbf{x}) dx = 0 \text{ for } k = 1 \dots M \tag{2.4}$$



In the Galerkin finite element method, the weighting functions use the same family of interpolating polynomials as the trial function [29]:

$$v_k(\mathbf{x}) = \sum_{i=1}^N \psi_{k,i}(\mathbf{x}) \quad (2.5)$$

It is possible to define a single interpolating polynomial to span the entire domain. However, this would be highly inaccurate if using a low-order polynomial and very high-order interpolating polynomials can become wildly oscillatory over large domains [30]. Instead, the domain is discretized into multiple subdomains by a mesh and the trial and weighting functions are defined to be piecewise continuous over the mesh elements. The interpolating polynomials range from zero outside of their specific mesh element to one at the mesh element nodes:

$$\psi_{k,i}(\mathbf{x}) = \begin{cases} 1 & \text{on node } i \text{ of } \Omega_k \\ 0 & \text{outside of } \Omega_k \end{cases} \quad (2.6)$$

thus the approximate solution is continuous across mesh element nodes. For further illustration see figure 2.1.

Inserting these into eqn. (2.4) results in a system of equations that can be solved for the coefficient values that minimize the residual on each mesh element:

$$\int_{\Omega} \sum_{i=1}^N \psi_{k,i} [a_i \nabla^2 \psi_i + f] dx = 0 \text{ for } k = 1 \dots M \quad (2.7)$$

The finite element method offers several advantages over other continuum-based numerical methods. It is preferable to use high-order numerical schemes—when computationally feasible—to increase the potential accuracy of the final simulation result. In the finite element method, polynomial interpolants are confined within single mesh elements. Thus, the order of the scheme can be increased simply by increasing the order of the individual polynomial interpolants in the trial and weighting functions and can even be increased separately in different mesh elements. In contrast, the finite difference method and finite volume method would require modified stencils or flux approximations that extend across multiple mesh nodes or elements and are challenging to use on unstructured meshes. For the same reason, the finite element method easily handles irregular or curved domains. Boundary terms, particularly flux-type boundary conditions, arise naturally from the finite element method formulation. They inherently satisfy the governing equations and

maintain the same accuracy order as the main finite element scheme. In contrast, boundary conditions must be added to the finite difference and finite volume methods after the governing equations have been discretized—often in a somewhat ad-hoc manner—and are not necessarily consistent with the scheme in the bulk domain. Finally, the finite element method can easily couple multiple governing equations over the same domain through the use of mixed finite element spaces. This is advantageous for computational multiphysics applications [10].

However, there are also shortcomings. Part of the unpopularity of the finite element method comes from how challenging it is to implement. It is also computationally expensive compared to other numerical methods and has high memory requirements, especially in three dimensions. Most significantly, the finite element method is not locally conservative and requires additional stabilization terms when used for convection-dominated problems. For this reason, fluid flow simulations have historically been dominated by the finite volume method [10].

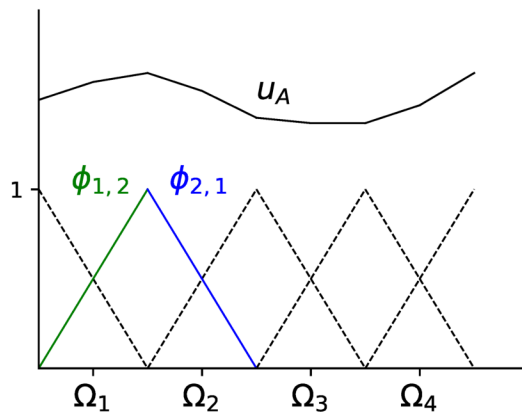


Figure 2.1: Example finite element method piecewise continuous trial and weighting functions. In one dimension each mesh element  $\Omega_k$  has two nodes and thus two interpolating polynomials.

### 2.2.2 The Discontinuous Galerkin Method

The discontinuous Galerkin method is a modification of the finite element method. Instead of using piecewise continuous polynomial interpolants, the trial and weighting functions are allowed to be discontinuous at mesh element edges as shown in figure 2.2. With

this, the discontinuous Galerkin method can enforce flux balances between mesh elements, generalizing the concepts of numerical fluxes and slope limiters from the finite volume method to the finite element method [11].

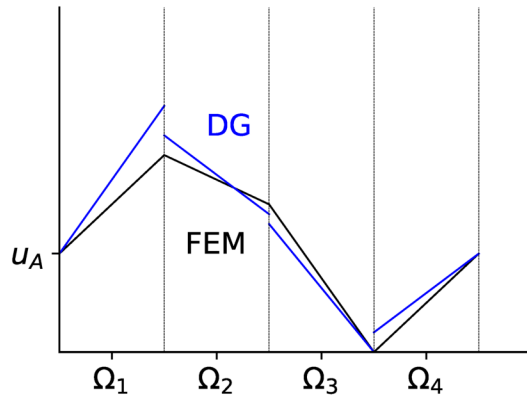


Figure 2.2: Example comparison of the trial functions of the finite element method (FEM) and the discontinuous Galerkin method (DG).

To see the benefits of this approach, consider the general hyperbolic problem:

$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{a}u) = 0 \quad (2.8)$$

$$u(\mathbf{x}, t = 0) = u_0(\mathbf{x}) \quad (2.9)$$

defined on some domain  $\Omega$  for some known velocity field  $\mathbf{a}$  and appropriate boundary conditions. Insert a trial function  $u_A$  to form the residual and average with a weighting function  $v$ :

$$\int_{\Omega} v \frac{\partial u_A}{\partial t} dx + \int_{\Omega} v \nabla \cdot (\mathbf{a}u_A) dx = 0 \quad (2.10)$$

Now apply the tensor product rule and the divergence theorem to the second term to form a boundary integral:

$$\int_{\Omega} v \frac{\partial u_A}{\partial t} dx - \int_{\Omega} u_A \mathbf{a} \cdot \nabla v dx + \int_{\Omega} \nabla \cdot (\mathbf{a}u_A v) dx = 0 \quad (2.11)$$

$$\int_{\Omega} v \frac{\partial u_A}{\partial t} dx - \int_{\Omega} u_A \mathbf{a} \cdot \nabla v dx + \int_{\Gamma} \mathbf{n} \cdot \mathbf{a}u_A v ds = 0 \quad (2.12)$$

where  $\Gamma$  denotes the boundary of the domain.

In the standard finite element method, the third term of eqn. (2.12) would be replaced by known values of flux-type boundary conditions. However, as fluxes can only be balanced over the entire domain the scheme will only be globally conservative. The scheme will also have significant stability issues if the exact solution contains discontinuities such as shock waves. Generally the standard Galerkin finite element method is unsuitable for hyperbolic equations and artificial diffusion methods or the streamline-upwind Petrov Galerkin finite element method must be used instead [31].

Now consider the discontinuous Galerkin method formulation. Since the discontinuous Galerkin method uses trial and weighting functions that are discontinuous across mesh element edges all domain integrals must be rewritten as sums of integrals over individual mesh elements  $\mathcal{K}$ :

$$\sum_{\mathcal{K} \in \Omega} \int_{\mathcal{K}} \left( v \frac{\partial u_A}{\partial t} - u_A \mathbf{a} \cdot \nabla v \right) dx + \sum_{\mathcal{K} \in \Omega} \int_{\partial \mathcal{K}} \mathbf{n} \cdot \mathbf{a} \hat{u}_A v ds = 0 \quad (2.13)$$

The third term of eqn. (2.13) can be further rewritten to account for each mesh element edge having two facets ( $\mathcal{F}$ ). These facets will be denoted by (+) and (-) and have normals which are parallel and in opposite directions:

$$\sum_{\mathcal{K} \in \Omega} \int_{\mathcal{K}} \left( v \frac{\partial u_A}{\partial t} - u_A \mathbf{a} \cdot \nabla v \right) dx + \sum_{\mathcal{F} \in \Omega} \left( \int_{\mathcal{F}^+} \mathbf{n} \cdot \mathbf{a} \hat{u}_A v ds + \int_{\mathcal{F}^-} \mathbf{n} \cdot \mathbf{a} \hat{u}_A v ds \right) = 0 \quad (2.14)$$

$$\sum_{\mathcal{K} \in \Omega} \int_{\mathcal{K}} \left( v \frac{\partial u_A}{\partial t} - u_A \mathbf{a} \cdot \nabla v \right) dx + \sum_{\mathcal{F} \in \Omega} \int_{\mathcal{F}} (\mathbf{n}^+ \cdot \mathbf{a}^+ \hat{u}_A^+ v^+ + \mathbf{n}^- \cdot \mathbf{a}^- \hat{u}_A^- v^-) ds = 0 \quad (2.15)$$

Also note that the trial function now has no single value at mesh element edges; the value of the trial function as evaluated within one mesh element containing an edge may be different from the value evaluated within the other mesh element that shares said edge. For further illustration, refer back to figure 2.2. Since a single value is needed to evaluate surface integrals over mesh edges, the trial function must be replaced by a numerical flux  $\hat{u}_A$  in these integrals. The form of the numerical flux can be chosen to ensure a flux balance across each mesh element, making the scheme locally conservative. With the appropriate choice of numerical flux (generally an upwinding scheme to enforce an appropriate direction of information flow and ensure a well-posed problem) the scheme is also stable and non-oscillatory [31]. Discontinuities in the solution can of course be captured by discontinuities in the trial function.

The discontinuous Galerkin method is very popular for hyperbolic equations for its stability and ability to capture shock dynamics with minimal artificial diffusion [11]. However, the discontinuous Galerkin method is also suitable for differential equations with

continuous solutions; the discontinuous Galerkin method trial functions are piecewise discontinuous, but these discontinuities will be on the order of machine precision when the exact solution is continuous. For example, Cockburn *et al* have shown the use of the discontinuous Galerkin method for the Stokes [32], Oseen [33], and incompressible Navier-Stokes equations [34]. In these cases, the discontinuous Galerkin method provides the additional benefit of enabling the use of HDiv-conforming finite elements to exactly enforce the incompressibility constraint [34]. The discontinuous Galerkin method can also be used for elliptic problems. See, for example, work by Arnold *et al* [35]. For full derivations of the discontinuous Galerkin formulations of the models used in `OpenCMP` see Appendix A.

Beyond its numerical advantages, the discontinuous Galerkin method has computational benefits compared to the standard finite element method. Mesh refinement is simplified by the lack of continuity across mesh element edges. In particular, hanging nodes—nodes which intersect mesh element edges—can be treated the same way as non-hanging nodes. This is extremely useful for adaptive mesh refinement of structured Cartesian grids, as shown in figure 2.3. Polynomial refinement is also easy as communication at mesh element edges is the same regardless of the order of the mesh element interpolant. Finally, the discontinuous Galerkin method is simple to parallelize since the mass matrix is block diagonal and the block size is dictated solely by the number of degrees of freedom on each mesh element. However, the discontinuous Galerkin method does have drawbacks: greater mathematical complexity in the formulation of the weak form, greater complexity in the numerical implementation to handle facet integrals, and increased memory requirements due to the significantly higher number of degrees of freedom [11].

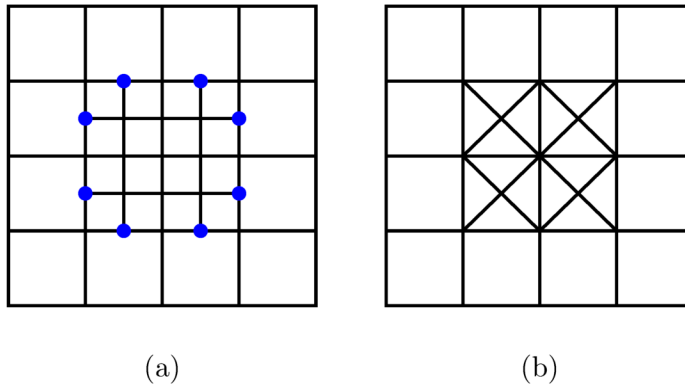


Figure 2.3: Adaptive mesh refinement with (a) hanging nodes (highlighted in blue) and (b) non-hanging nodes. The use of non-hanging nodes requires the addition of triangular mesh elements to the structured quadrilateral mesh.

## 2.3 The Diffuse Interface Method

The numerical methods described in the previous section all rely on discretizing the governing equations and minimizing the residual at select locations in the simulation domain. To do this, the simulation domain is represented by a mesh of simple polygonal elements as shown in figure 2.4. The mesh conforms perfectly to the simulation domain at the mesh nodes, but not generally along the mesh element edges. Mesh refinement is needed for sufficient conformance as to not significantly impact simulation accuracy, particularly in areas of high curvature [36].

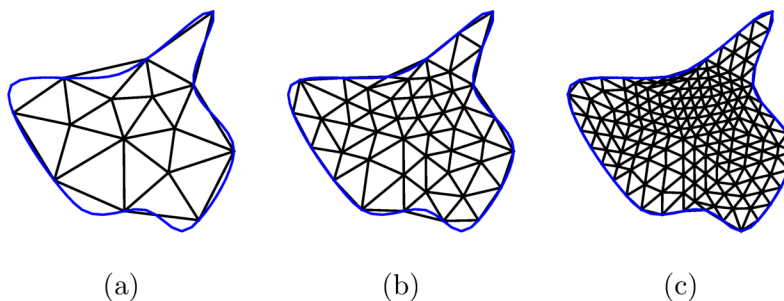


Figure 2.4: An example simulation domain (blue) meshed by increasingly refined unstructured conformal meshes. Conformance of mesh elements to the simulation domain boundary improves with increasing refinement.

Figure 2.4 shows an unstructured triangular mesh. Structured meshes and meshes with quadrilateral/hexahedral mesh elements are also possible (see figure 2.5).

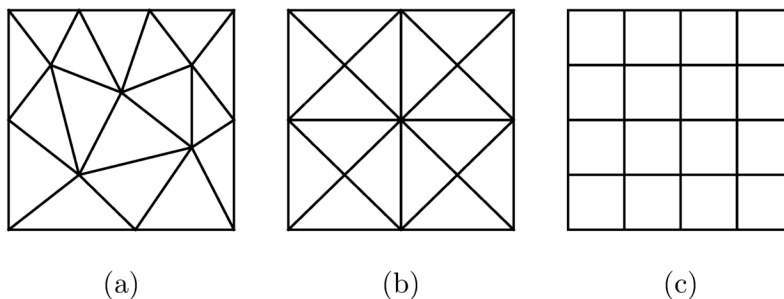


Figure 2.5: Examples of (a) an unstructured triangular mesh, (b) a structured triangular mesh, and (c) a structured quadrilateral mesh.

Unstructured triangular/tetrahedral meshes are commonly used in computational multiphysics as the complex geometries inherent to real-world problems are challenging to conformally mesh with structured and/or quadrilateral/hexahedral meshes [37]. However, this forgoes the many benefits of structured quadrilateral/hexahedral meshes. Structured meshes have simple connectivity structures so are easier and less memory intensive to store than unstructured meshes. Simulations are less computationally intensive as structured meshes produce banded matrices which are easy to solve. Additionally, structured meshes are highly amenable to GPU acceleration [5]. Quadrilateral/hexahedral mesh elements improve simulation stability and numerical accuracy if aligned with the dynamics of the solution field, for example, the direction of flow in fluid dynamics simulations. They can also be stretched in this significant direction without skewing their shape, allowing coarser meshes without sacrificing mesh quality [4].

Beyond the forced reliance on unstructured meshes, generation of conformal meshes of complex geometries is very time and labour intensive. Automated mesh generation algorithms exist, particularly for unstructured triangular/tetrahedral meshes. However, the initial outputs of these algorithms are typically low quality and time must be spent identifying and removing skewed, inverted, or sliver elements. Time must also be spent selectively refining the mesh in regions expected to have high solution gradients, such as near walls. For best mesh quality it is often preferable for the user to manually block out different regions of the geometry for automated structured mesh generation then improve the mesh at the interfaces of these regions. This labour and time investment makes up the majority of the total simulation effort [3, 4].

The diffuse interface method offers an alternative to conformally meshing complex geometries. It is a type of immersed boundary method where the complex geometry is enclosed in a structured non-conforming mesh then mapped to the nodes of said mesh by a phase field. See figure 2.6 for an example. The phase field  $\phi$  is a scalar field, equal to one on mesh elements inside of the complex geometry and zero on mesh elements outside of the complex geometry, and which varies smoothly from zero to one at the boundary of the complex geometry. The sharp boundary of the complex geometry can be approximated by the level set  $\phi = 0.5$  while the diffuse boundary region of finite thickness is found where  $|\nabla\phi| > 0$ .

When the finite element method is used, the weak form of the original conformal mesh problem can be reformulated for the diffuse interface method using the following integral

identities [38]:

$$\int_{\Omega} A dx = \int_{\kappa} AH dx \approx \int_{\kappa} A\phi dx \quad (2.16)$$

$$\int_{\Gamma} B ds = \int_{\kappa} \delta_{\Gamma} B dx \approx \int_{\kappa} B|\nabla\phi| dx \quad (2.17)$$

$$\mathbf{n} \approx \frac{-\nabla\phi}{|\nabla\phi|} \quad (2.18)$$

where  $\Omega$  is the complex geometry with boundary  $\Gamma$  and  $\kappa$  is the enclosing domain.  $H$  is the Heaviside function,  $\delta$  is the Dirac delta function, and  $A$  and  $B$  are generic fields. Diffuse interface method formulations of all of the models used in `OpenCMP` are given in Appendix B.

The main benefit of any immersed boundary method is the ability to use structured quadrilateral/hexahedral meshes for any complex geometry, avoiding the stability issues and heavy time and labour requirements of unstructured meshes. However, the lack of mesh conformance to the complex geometry boundary negatively impacts simulation accuracy. Boundary conditions end up smeared over multiple mesh elements, curved boundaries gain a step-like appearance, and the apparent size of the complex geometry may change. Prior work has alleviated this issue either by modifying the integration schemes within mesh elements containing the complex geometry boundary, as with the cut cell methods used by Nguyen *et al* [39], or by significantly refining the mesh near the complex geometry boundary [38]. However, both of these tactics significantly increase the computational complexity of the simulation, rendering the savings in mesh complexity moot.

This thesis will instead follow the work of Monte *et al* [8] and focus on the potential of the diffuse interface method to speed up inherently low accuracy simulations. A significant use of computational multiphysics is design screening and optimization. This involves simulating many different designs or variations of a design to narrow down on the best few. High accuracy is not necessary as the best designs can be further evaluated through more extensive simulation or physical prototyping. However, increasing simulation speed would allow more potential designs to be evaluated or speed up the overall design process. Thus, as long as the diffuse interface method is sufficiently accurate to distinguish between different designs on select important performance parameters, its time savings compared to standard conformal meshing can be valuable.



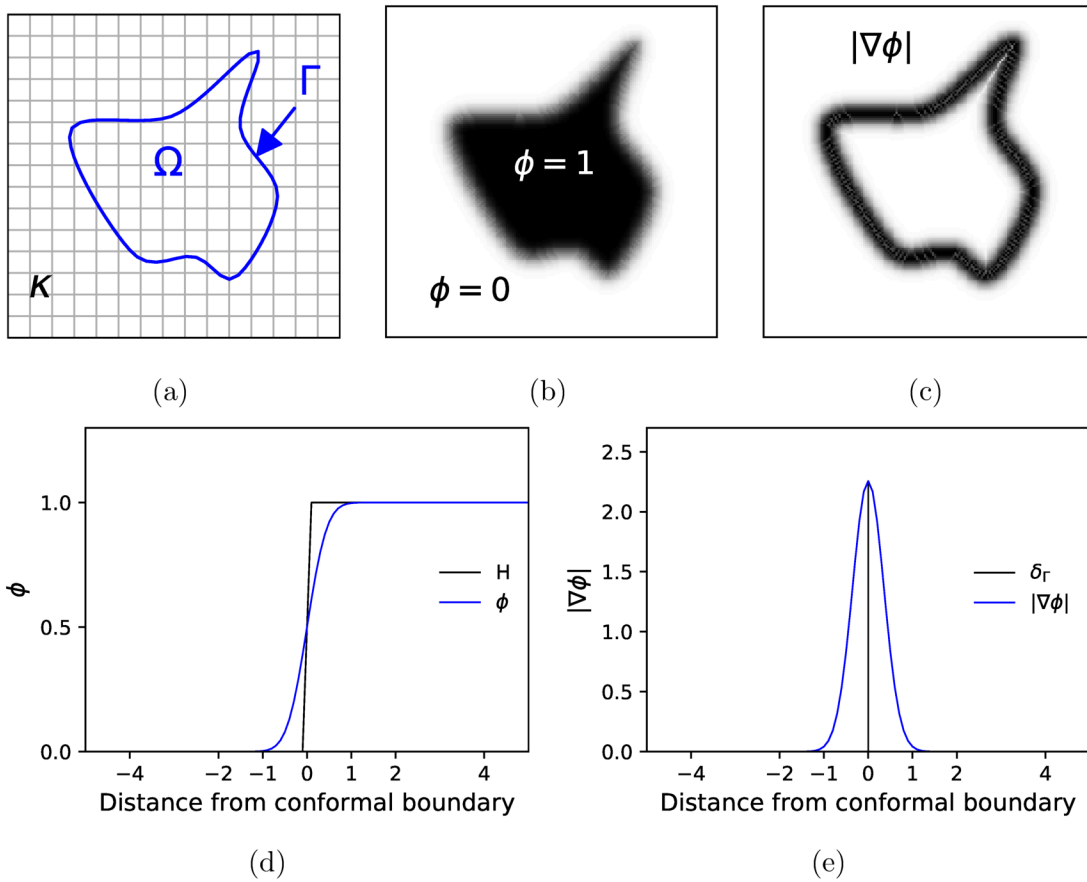


Figure 2.6: Example of (a) a complex geometry meshed by a non-conforming mesh (mesh coarsened for visibility), (b) the phase field approximation of the geometry, and (c) the phase field approximation of the complex geometry boundary. Also the approximation of (e) the Heaviside function by the phase field and (f) the Dirac delta function by the magnitude of the gradient of the phase field.

# Chapter 3

## Related Work

Computational multiphysics is a broad field, even when restricted to chemical engineering-relevant applications. As such, existing computational multiphysics software packages vary widely in their purpose, capabilities, interface, and numerical implementation. However, all computational multiphysics software intended for use beyond its original developers must carefully consider its overall usability. This chapter will describe some key design decisions that affect the potential use cases, likely audience, and user friendliness of a computational multiphysics software package, then review the choices made by popular existing packages. Raw performance is not discussed as it is heavily application-specific and difficult to compare in a meaningful way across different types of computational multiphysics software packages.

### 3.1 Design Considerations

#### 3.1.1 Models and Numerical Methods

The models available in a computational multiphysics software package constrain its potential use cases, while the numerical methods implemented primarily affect performance. As mentioned in section 2.1, most chemical engineering applications of computational multiphysics center on fluid dynamics, heat and mass transfer, and chemical reaction. Thus, the most relevant models are the following [4]:

- The incompressible Navier-Stokes equations and simplifications like Stokes flow
- Heat conduction

- Conjugate heat transfer
- Turbulence models: Reynolds-averaged Navier-Stokes, large eddy simulation, and sometimes direct numerical simulation
- Multi-component flow
- Multi-phase flow
- Reacting flow
- Particle tracking
- Fluid-structure interaction
- Rotating geometries

It should be noted that many of these models come from fluid dynamics and it is common for computational fluid dynamics software packages to feature most, if not all, of the models listed above. The benefit of computational multiphysics software packages, which further include models for structural mechanics, acoustics, electromagnetics and more, is the ability to extend to multidisciplinary projects [2].

Section 2.2 compared common spatial discretization schemes for continuum model simulations. When discussing the numerical methods in a computational multiphysics software package, time discretization schemes are also of interest as are, more broadly, techniques for improving computational efficiency such as the inclusion of adaptive time-stepping or algorithms for adaptive mesh or polynomial order refinement. Similarly, the specific formulations of the model governing equations, for instance, the choice to use the primitive variable form of the incompressible Navier-Stokes equations instead of the pressure-Poisson form [40], or large-scale reformulations such as the use of the diffuse interface method. Some computational multiphysics software packages have arisen to showcase specific numerical methods. For example, the adjoint solvers in SU2 [41] or the inclusion of the diffuse interface method in OpenCMP. However, many users, especially those not active in numerical methods research, are primarily interested in the overall performance and comprehensiveness of the computational multiphysics package.

### 3.1.2 Location in the Simulation Workflow

A major design consideration that governs the overall use of a computational multiphysics software package is how it fits into the simulation workflow. Tu *et al* [4] suggest breaking the general simulation workflow into three main steps:

1. Pre-processing
  - 1.1. Geometry creation
  - 1.2. Mesh generation

2. Solver
  - 2.1. Model specification
  - 2.2. Solver specification
  - 2.3. Numerical solve
3. Post-processing
  - 3.1. Parameter computation
  - 3.2. Error analysis
  - 3.3. Visualization

Step one, pre-processing, is a necessary component of the simulation workflow, but is often handled by third-party software instead of the main computational multiphysics software package. Real-world applications generally involve complex geometries that are best constructed with dedicated computer-aided design (CAD) software. Likewise, the moderate mesh generation capabilities of most computational multiphysics packages are not sufficient to produce high quality meshes of these complex geometries. However, the most comprehensive computational multiphysics software packages integrate directly with CAD software and have extensive mesh generation and optimization algorithms. These high performance algorithms become an additional selling point of the computational multiphysics software and minimizing dependence on third-party software significantly increases user friendliness.

Step two, the models and solver, is a necessary component of any computational multiphysics software package. This includes the ability to specify model governing equations, boundary conditions, and physical properties; the solver and solve criteria such as convergence tolerances; and the numerical methods used for the actual simulation. Different levels of control over the numerical implementation suit different audiences; researchers may wish to insert custom models and solvers, while users in industry may prefer a small set of default solvers with validated performance.

Step three, post-processing, is the final analysis of simulation results. When used in the engineering design process, simulations are typically conducted to estimate key design parameters. For example, evaluation of a heat exchanger design may concentrate on the log mean temperature difference for different fin spacing. Conversely, verification of a new numerical method or model may require an error comparison of the full solution fields against a known reference solution. Most computational multiphysics software packages include the ability to compute user-defined metrics and save simulation results to file. Visualization, of these metrics or of the solution fields, is generally dependent on the software user interface. Computational multiphysics software packages with graphical interfaces can offer visualization capabilities, but otherwise visualization must be relegated to third-party software. This is again a significant factor in the user friendliness of the package.

The simulation workflow comprises one single simulation run. However, typical use of computational multiphysics software packages can involve many repetitions of the same simulation to identify a mesh-independent solution, iterate on a design, or test the effects of minor environmental variations. Tools for design optimization, parametric design exploration, and for automating common workflows can improve utility beyond the basic simulation workflow. These tools are, for the most part, found only in industry-focused commercial packages.

### 3.1.3 User Interface

There are three main types of user interface used by computational multiphysics software packages: script-based user interfaces, configuration file-based user interfaces, and graphical user interfaces. The type of user interface helps dictate the audience of the computational multiphysics software package. It can also affect the software capabilities such as the ability to visualize results.

#### Script-Based User Interface

A script-based user interface is the most basic type of user interface. The computational multiphysics software package acts simply as a library of numerical methods for the user to use while coding the simulation workflow.

An example, which uses the `NGSolve` finite element solver library [13], is shown in figure 3.1. Similar to if the user were writing their own finite element method code from scratch, the user must construct the weak form for their model, apply boundary conditions, and solve the resulting system of equations to minimize the residual. However, instead of implementing their own finite element spaces and solvers the user imports them from the computational multiphysics library. Thus, the user has easy access to high performance solvers and can focus their time on model development.

A script-based user interface provides the highest degree of user control over the simulation. The user has complete control over the model governing equations and how they are formulated. They control the numerical methods used, such as the type of finite element used, and can generally choose from a variety of solvers and control solver tolerances and convergence criteria. Script-based user interfaces are well-suited to automation; the user can simply wrap their main simulation code in a loop that controls input values. Additionally, any text-based interface can be added to version control software to track changes

```

from ngsolve import *
from netgen.geom2d import unit_square
import netgen.gui

# Create the mesh.
mesh = Mesh(unit_square.GenerateMesh(maxh=0.1))

# Define the finite element space.
fes = H1(mesh, order=2, dirichlet='top|bottom|left|right')
u = fes.TrialFunction()
v = fes.TestFunction()

# Define the boundary condition and source function.
g = 1 + x*x + 2*y*y
f = -6

# Construct the weak form as a bilinear and linear form.
a = BilinearForm(fes, symmetric=True)
a += Grad(u) * Grad(v) * dx
a.Assemble()

L = LinearForm(fes)
L += f * v * dx
L.Assemble()

# Define a gridfunction to hold the solution and apply the
# Dirichlet boundary condition to it.
sol = GridFunction(fes)
sol.Set(g, definedon=mesh.Boundaries('top|bottom|left|right'))

# Solve the system of equations.
inverse = a.mat.Inverse(freedofs=fes.FreeDofs())
residual = L.vec.CreateVector()
residual.data = L.vec - a.mat * sol.vec
sol.vec.data += inverse * residual

# Plot the solution.
Draw(sol)

```

Figure 3.1: An example NGSolve [13] script to solve the Poisson equation.

to simulation parameters and model formulations for easy replication. Overall, computational multiphysics software packages with script-based user interfaces are highly suited to research applications, particularly those involving custom models or novel numerical methods.

However, such a high degree of user control requires a correspondingly high degree of user experience. The user must understand exactly how to implement their governing equations including, for instance, how to incorporate time discretization, whether stabilization terms are necessary, and what finite element spaces are appropriate to use. The user must also have a decent amount of programming experience in the language of their chosen computational multiphysics software package. A script-based interface is particularly prone to inefficiencies due to poor coding practices and misuse of the computational

multiphysics software. Finally, the computational multiphysics software only guarantees the performance of its numerical methods, the user must still verify their overall code implementation and validate their model formulation themselves. There is a high potential for nonphysical results due to user errors.

## Configuration File-Based User Interface

A configuration file-based user interface goes a step beyond a script-based user interface. Common models are implemented in the code back-end and the user simply chooses their model of interest and specifies relevant values for the physical properties, boundary conditions, etc in an input file. Then the simulation is run through the command line by pointing the computational multiphysics software to the relevant configuration file.

An example from the SU2 [41] computational fluid dynamics software package is shown in figure 3.2. Note that only a selection of the full set of configuration file parameters is shown. Also note that this is a plain .txt file, not a script. In this example, the user is able to simulate the Euler equations simply by specifying "EULER" as the "SOLVER" parameter; the relevant governing equations are already implemented.

```

% ----- DIRECT, ADJOINT, AND LINEARIZED PROBLEM DEFINITION -----%
%
% Physical governing equations (EULER, NAVIER_STOKES,
%                               WAVE_EQUATION, HEAT_EQUATION, FEM_ELASTICITY,
%                               POISSON_EQUATION)
SOLVER= EULER
%
% Mathematical problem (DIRECT, CONTINUOUS_ADJOINT)
MATH_PROBLEM= DIRECT

% ----- COMPRESSIBLE AND INCOMPRESSIBLE FREE-STREAM DEFINITION -----%
%
% Mach number (non-dimensional, based on the free-stream values)
MACH_NUMBER= 0.8

% ----- BOUNDARY CONDITION DEFINITION -----%
%
% Marker of the Euler boundary (NONE = no marker)
MARKER_EULER= ( airfoil )
%
% Marker of the far field (NONE = no marker)
MARKER_FAR= ( farfield )

% ----- LINEAR SOLVER DEFINITION -----%
%
% Linear solver for implicit formulations (BCGSTAB, FGMRES)
LINEAR_SOLVER= FGMRES
%
% Preconditioner of the Krylov linear solver (JACOBI, LINELET, LU_SGS)
LINEAR_SOLVER_PREC= ILU

% ----- INPUT/OUTPUT INFORMATION -----%
% Mesh input file
MESH_FILENAME= mesh_NACA0012_inv.su2

```

Figure 3.2: Portion of an example SU2 configuration file for flow past an airfoil (from [42] with modifications).

Pre-implemented models are the main benefit of a configuration file-based user interface over a script-based user interface. Beginner users in particular can quickly begin running simulations without an in-depth knowledge of the governing equations and numerical methods. The developers of the computational multiphysics software package will have validated the model performance and correctness for the users. Computational efficiency may also be higher since all code is under the control of the software developers. Yet, most computational multiphysics software packages with configuration file-based user interfaces still provide access to their code back-end. It is typically still possible for experienced users to treat the packages as script-based packages and modify models or implement their own models. Configuration file-based user interfaces are still suited to automation by external scripts that modify the configuration files and run the software package through the command line. These are also still text-based interfaces and still amenable to version control. In many ways, configuration file-based user interfaces can be considered the happy medium suitable for beginner and experienced users alike.

However, there are still disadvantages. Pre-implemented models remove the potential for errors in the model implementation, but users must still understand which models, boundary conditions, and ranges of parameter values are physical for their particular use case. Configuration file-based interfaces are unsuitable for meshing and results visualization without an accompanying graphical interface. They also require users to be familiar with the command line. Finally, configuration files can become large and unwieldy in an effort to provide user control over all relevant simulation parameters. For example, the SU2 configuration file template, which includes all possible parameters, is 1686 lines long [43]. OpenFOAM® [20], which takes the opposite approach and splits the simulation information across multiple configuration files, is able to limit individual configuration file sizes to several hundred lines, but then requires at least 11 separate configuration files for one single simulation.

## Graphical User Interface

A graphical user interface is the most sophisticated type of user interface. Users interact with the computational multiphysics software package through menus, buttons, and dialog boxes while windows are used to visualize simulation progress and results.

Figure 3.3 shows an example of the COMSOL Multiphysics® [12] graphical user interface. The tree in the leftmost panel maps out the various components of the simulation including the geometry, mesh, physical properties, governing equations, solver, and results visualization. Clicking on any item in the tree opens it in the middle panel and gives access



to dialog boxes and drop-down menus to set parameter values. The rightmost panel is used for visualization and the top menu bar gives access to additional features.

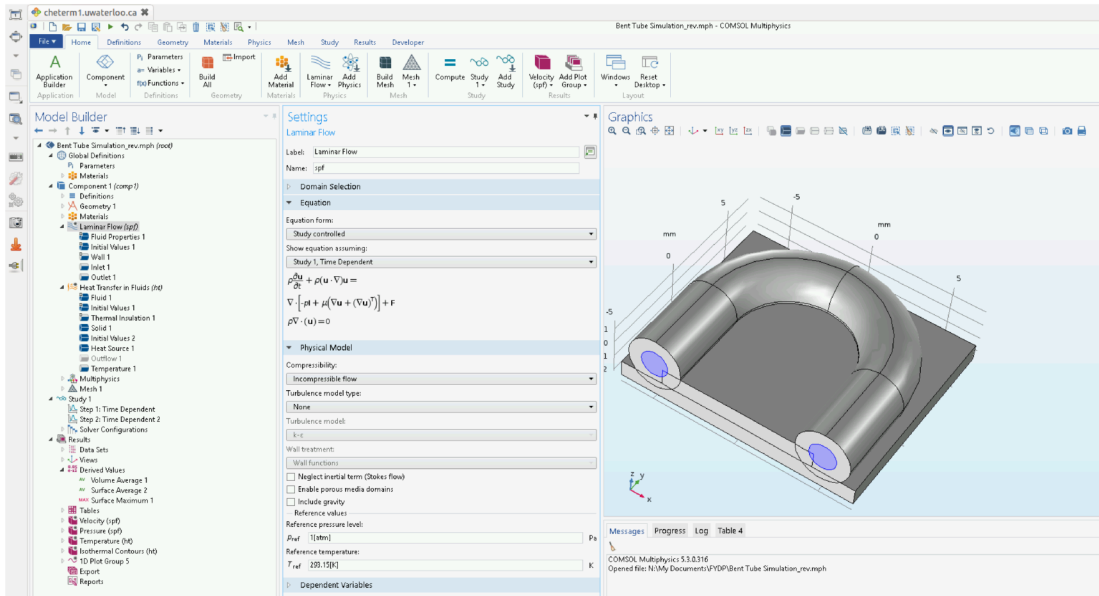


Figure 3.3: An example of the COMSOL Multiphysics® [12] graphical user interface.

Graphical user interfaces are the most user friendly type of user interface. Like configuration file-based user interfaces, all models are pre-implemented. Beyond this, there are typically restrictions on the combinations of specific models, boundary conditions, and the range of values of physical properties to those which are expected to be physical. This is particularly helpful for beginner users. The graphical interface allows meshing and results visualization to be built directly into the computational multiphysics software package. Furthermore, most computational multiphysics software packages with graphical user interfaces also include a command line user interface for experienced users to access additional functionality or for automation of simulation runs.

However, graphical user interfaces by necessity limit user control to predefined options and it is less common for users to have access to the code back-end. Thus, graphical user interfaces are only suitable for computational multiphysics software packages with very comprehensive sets of models or for users with very common use cases. Automation is generally limited to built-in functionality, though this is quite extensive for some computational multiphysics software packages. It is more-or-less impossible to track inputs to a graphical interface with version control software. It is also generally impossible to read saved configurations without loading them into the original computational multiphysics

software. From a developer perspective, creation of a graphical user interface is a much larger task than a configuration file-based user interface. Graphics are also computationally intensive compared to command line output.

## Programming Language

A related consideration is the programming language used by the code back-end. This impacts both the performance of the computational multiphysics software package and its ease of use if users have access to the code back-end. For reasonable computational speed the base code, primarily the solver, must use a compiled language. C++ [44] is a common choice, see for example deal.II [45], OpenFOAM® [20], and MOOSE [46]. Some packages, like FreeFEM++ [47], even implement their own programming languages. However, this necessitates users learning said programming language. A recent trend is to include a wrapper to a high-level interpreted language. This is predominantly for computational multiphysics software packages with script-based user interfaces. For example, FEniCS [16] and NGSolve [13] both have extensive Python [48] wrappers to their C++ back-ends. A wrapper offers a simpler more readable language for major user interaction while retaining most of the performance benefits of an entirely compiled package. A related option is to write the entire computational multiphysics software package in a language like Julia [49]. Julia is a just-in-time compiled language with easy to read syntax similar to high-level languages. It retains the performance benefits of a C++-based package and the usability of a Python-based interface without the need to maintain code in multiple different languages. However, Julia is quite new and remains relatively unknown.

### 3.1.4 Closed- or Open-Source

Though it does not affect the mechanics of running simulations with a computational multiphysics software package, the choice to make the software either closed- or open-source affects the type of user and use cases.

Closed-source means the source code is proprietary and not viewable by users or the general public. Generally closed-source software is also commercial software, *ie* must be paid for. Open-source means the source code is freely available to the general public. Many open-source packages are free and distributed under licenses that allow anyone to use or modify their source code. Some open-source computational multiphysics software packages also provide commercial services such as consulting and technical support.

Keeping software closed-source is mainly advantageous to its developers, as it allows them to profit directly off of the software. Users are disadvantaged as the source code is hidden and cannot be examined for specific model implementation details. Undisclosed assumptions may affect the validity of a model setup, especially for unconventional applications. Exact replication of results is also impossible as the exact implementation is unknown. Exact replication may not even be possible between different versions of the same software package which can become highly problematic as older versions are deprecated and become no longer available. Users are unable to modify models or implement custom models beyond what is possible through the user interface, so closed-source computational multiphysics software packages may be unsuitable for research applications, particularly if they lag behind the most recent work in the field. Finally, the software licenses are typically very expensive. However, the ability to profit off of the software also incentivizes continued support and development. Commercial closed-source computational multiphysics software packages are the most comprehensive computational multiphysics software packages available. They are also the most user friendly and have among the most readily available and extensive official support and documentation.

Open-source software is often developed primarily by volunteers. It is highly susceptible to being abandoned by the original developers and support and documentation often falls to the community of users. Software quality can also be highly variable. For computational multiphysics software packages in particular, open-source software is often highly specialized for specific applications. It is rarely as comprehensive as closed-source alternatives, so users must often integrate multiple different packages to cover the entire simulation workflow. However, the benefit is complete knowledge of and ability to modify the code implementation. This is essential for users interested in developing or replicating cutting-edge models and numerical methods.

## 3.2 Current Options

Many computational multiphysics software packages already exist. The following sections will describe some of the most popular, grouped by their general structure and intended audience. Computational fluid dynamics software packages are also included since, as discussed in section 3.1.1, they often offer sufficiently comprehensive feature sets for chemical engineering applications.

### 3.2.1 Commercial Software Packages

The first category of existing software is commercial software packages which, as the name implies, are commercial closed-source packages. The objective of these packages is to be comprehensive and intuitive. They encompass the entire simulation workflow. They are also often integrated into a broader suite of simulation tools including CAD generation and automated design optimization. This can include tools for additional physics—acoustics, electromagnetics, and structural mechanics—if the package is not already a true computational multiphysics software package. The user experience is primarily through extensive graphical interfaces, with optional command line customization for experienced users.

Some of the major commercial computational multiphysics software packages are COMSOL Multiphysics<sup>®</sup> [12], ANSYS Fluent<sup>®</sup> [50], ANSYS CFX<sup>®</sup> [51], and Simcenter STAR-CCM+ [52]. COMSOL Multiphysics<sup>®</sup> is one of the few commercial packages based exclusively on the finite element method, including support for the discontinuous Galerkin method. It is known for extreme user friendliness, even by the standards of closed-source software. Its graphical user interface is notable for the built-in interpreter, which allows for custom user-specified model equations [12]. ANSYS Fluent<sup>®</sup> and ANSYS CFX<sup>®</sup> are two of the many simulation software packages offered by Ansys Inc. and comprise its main computational fluid dynamics software. ANSYS Fluent<sup>®</sup> uses the traditional finite volume method while ANSYS CFX<sup>®</sup> uses the vertex-centred finite volume method. They are also optimized for different applications; ANSYS Fluent<sup>®</sup> is a general purpose computational fluid dynamics software package while ANSYS CFX<sup>®</sup> is recommended for turbomachinery simulations [50, 51]. Simcenter STAR-CCM+ is geared towards industry use and offers particularly extensive design optimization tools including automated design exploration, specialized algorithms for fast re-meshing of modified geometries, and stochastic analysis to estimate the performance effect of specific design factors. It also includes an automated simulation assistant to enforce company-defined simulation best practices [52].

### 3.2.2 Commercial Platforms and Toolboxes

The second category of existing software is commercial platforms and toolboxes. Like the previous category, these are commercial packages. However, instead of being fully closed-source, they are platforms built on top of pre-existing open-source computational multiphysics software. These platforms typically integrate many different open-source packages to provide a more comprehensive set of models than any one package and to support the full simulation workflow, including mesh generation and visualization. They provide a

graphical user interface for ease of use and their documentation and support services are often superior to those of the underlying open-source packages.

Two popular commercial platforms are `SimScale` [53] and `FEATool Multiphysics`<sup>™</sup> [54]. `SimScale` is built on the open-source software packages `OpenFOAM`<sup>®</sup> [20], `Code_Aster` [55], and `CalculiX` [56]. It offers commercial products for companies and a free community edition for students and hobbyists. `SimScale` also offers a cloud-based high performance computing system on which to run its simulation software. It caters to small companies and individual users who cannot afford to or do not wish to maintain their own computing infrastructure [53]. `FEATool Multiphysics`<sup>™</sup> provides both a graphical user interface and a script-based interface to `MATLAB` [57], `OpenFOAM`<sup>®</sup> [20], `SU2` [41], and `FEniCS` [16]. It offers simulation consulting services in addition to its software [54].

### 3.2.3 Open-Source Software Packages

The third category of existing software is open-source software packages. These are the open-source equivalent of *ex* `COMSOL Multiphysics`<sup>®</sup> [12] or `ANSYS Fluent`<sup>®</sup> [50]. They are meant to encompass the majority of the simulation workflow and be reasonably user friendly with predefined models and user interaction through a graphical user interface or configuration file-based interface. Most open-source software packages began as specialized software for specific applications, but the most popular have expanded to include a large range of computational multiphysics or computational fluid dynamics models.

`OpenFOAM`<sup>®</sup>/`FOAM-Extend` [20, 58, 59] is likely the most widely used open-source computational fluid dynamics software package. It is based on the finite volume method and supports the main simulation workflow, from meshing to visualization and post-processing of results. User interaction is through a configuration file-based interface and various unofficial graphical interfaces are also available [60]. `SU2` [41] is another common open-source computational fluid dynamics software package. It was originally developed to solve the Reynolds-averaged Navier-Stokes equations and is still predominantly used for aerodynamics applications. `SU2` is notable for its specialized adjoint solvers which provide highly computationally efficient calculations of solution field gradients. It also has extensive adaptive mesh refinement, mesh deformation, and shape optimization capabilities. `SU2` is mainly based on the finite volume method, but provides some finite element and discontinuous Galerkin solvers for multiphysics simulations [41].

`MOOSE` [46] and `Elmer` [15] are open-source multiphysics software packages. Both are based on the finite element method and support the full simulation workflow apart from mesh generation. `Elmer` is used primarily through its extensive graphical user interface

with an optional command line interface [15]. MOOSE requires more scripting than is typical of open-source software packages. The user constructs their model as a "kernel", calling and combining different pre-implemented physics within an input script. All remaining simulation parameters, such as boundary conditions and physical properties, are specified through configuration files. This user interface uses C++ [44], but there are several helper libraries based on Python [48]. To make up for its relatively steep learning curve, MOOSE offers possibly the most extensive set of models of any open-source simulation software and includes many tools for code testing and documentation [46].

SfePy [17] represents a subset of open-source simulation software packages meant for solving general partial differential equations, but not geared towards any specific physics. It provides broad support for generic partial differential equations formed from combinations of common integral terms. SfePy is less popular than multiphysics-specific packages, but can be a powerful tool for uncommon models. Beyond the finite element solver, only basic meshing and post-processing operations are supported, though SfePy can load third-party meshes and output results for post-processing in third-party software. User interaction is either through configuration files or through a script-based interface. A graphical interface is also provided for basic visualization [17].

### 3.2.4 Open-Source Solver Libraries

The fourth category of existing software is open-source solver libraries. These provide the numerical methods and solvers needed to run computational multiphysics simulations, though no pre-implemented models like the previously discussed software packages. User interaction is generally through a script-based interface. Open-source solver libraries are generally application agnostic; the solvers can be applied to any multiphysics model. Thus, though not user friendly, they can be a powerful tool, particularly for cutting-edge numerical methods research. This discussion will focus specifically on finite element solver libraries due to their relevance to OpenCMP.

FreeFEM++ [47] is one of the oldest finite element solver libraries, initially released as MacFEM in 1987. deal.II [45] and libMesh [61] are also among the oldest solver libraries still under active development, both released in the early 2000s. Of the three, deal.II is the most widely used. It was designed and optimized for computation speed and efficiency as exemplified by the choice to build it in C++. Significant work was also devoted to algorithms for adaptive mesh refinement and coarsening, including support for hanging nodes, though the mesh generation capabilities are limited. deal.II is notable for some of the most extensive documentation and tutorials of open-source computational fluid dynamics software [45].

A major innovation in finite element solver libraries was the introduction of unified form language [62] in the `FEniCS` finite element solver library [16]. This simplifies model definition by allowing the finite element method weak form to be passed into the solver in quasi-mathematical notation instead of the user manually assembling the mass matrix. `NGSolve` [13] is another finite element solver library that uses symbolic weak form notation, though not unified form language. `FEniCS` and `NGSolve` both also provide `Python` [48] wrappers to their `C++` back-ends to further improve ease of use. `NGSolve` is known for its strong support of the discontinuous Galerkin method and the hybrid discontinuous Galerkin method. It also integrates with the meshing software `Netgen` for mesh generation and a graphical interface for visualization [13].

`DUNE` [18] deserves special mention for its extensive capabilities, though it is more properly a framework of multiple solver libraries. It has core libraries (modules) which provide the basic code base and are maintained by the `DUNE` developers. Extensions provide additional functionality and support for specialized applications. `DUNE` has extensive support for grid adaptivity. It offers mesh and polynomial order refinement including efficient parallelization of adaptive mesh refinement [18]. The `DUNE-FEM` [19] extension module provides finite difference, finite volume, and finite element implementations for solving differential equations. `DUNE-PDELab` [14] provides model templates for `DUNE-FEM` and uses the `DUNE Python` wrapper to represent weak forms with unified form language.

A final example of note is `Gridap` [63]. `Gridap` is one of the few solver libraries written in `Julia` [49] and may be a trail blazer for a new class of finite element solver library.

### 3.2.5 Auxiliary Software

The final category comprises not computational multiphysics/computational fluid dynamics software but auxiliary software used to support or enhance computational multiphysics software packages.

`Gmsh` [64], `Netgen` [13], and `Salome` [65] are meshing software. `Gmsh` and `Salome` both include CAD engines for generation of complex geometries prior to meshing. `Netgen` supports importing CAD files but its own geometry creation capabilities are limited to simple primitives. All three include routines for geometry clean-up, mesh optimization, and mesh quality checks. `Gmsh` is an independent software, though many computational multiphysics software packages can import native `Gmsh` meshes. In contrast, `Netgen` and `Salome` are both integrated into broader open-source software ecosystems, `Netgen` with `NGSolve` [13] and `Salome` with `Code.Aster` [55] and related packages. `Netgen` and `Salome` can both be used for visualization of simulation results from their coupled computational multiphysics



software packages. **Salome** further provides extensive post-processing functionality through its graphical user interface.

**ParaView** [66] is possibly the most popular open-source software for data visualization and post-processing. Many computational multiphysics software packages support exporting results to its `.vtk` format. **ParaView** has also been integrated into several open-source software packages like `paraFoam` in **OpenFOAM**<sup>®</sup> [20].



# Chapter 4

## Overview of OpenCMP

This chapter provides an overview of the `OpenCMP` computational multiphysics software package: the finite element solver library on which it is built, its overall code framework, its user interface, and its simulation capabilities. `OpenCMP` is an open-source project released under the GNU LGPL license [67]. Its code repository is publicly available at <https://github.com/uw-comphys/opencmp> and documentation and examples are available on the website <https://opencmp.io/>.

### 4.1 Solver Back-end

The starting point of `OpenCMP` development was the choice of finite element solver library. Writing an implementation of the finite element method from scratch was deemed infeasible; it would be a very time consuming process and the final result would be inferior to existing libraries. On the other hand, using an existing library for the back-end of `OpenCMP` would allow effort to be focused on implementing multiphysics models and developing a user interface and general simulation framework. Existing finite element solver libraries are also already optimized for performance and validated for efficiency and accuracy. As such, `OpenCMP` is built on the open-source `NGSolve` finite element library [13].

One major benefit of `NGSolve` is its `Python` [48] wrapper and symbolic representation of finite element weak forms. As mentioned in section 3.1.3 and section 3.2.4, these elements significantly increase code readability. This speeds up the development of `OpenCMP`. Having readable code also aids experienced users who may wish to modify `OpenCMP` models or add their own models. `NGSolve` is available for all standard operating systems and has minimal

dependencies, allowing `OpenCmp` to have the same. It integrates with the `Netgen` meshing software and graphical interface [13] which, in the future, can allow `OpenCmp` to add mesh generation and results visualization capabilities. `NGSolve` also offers several performance benefits over other finite element solver libraries. It implements an extensive set of finite element spaces including those for fluid dynamics, structural mechanics, electromagnetics, and acoustics, so is suitable for true multiphysics simulations. `NGSolve` has the strongest support of existing finite element libraries for the discontinuous Galerkin method. It also offers a wide variety of state-of-the-art solvers and preconditioners including interfaces to `SciPy` [68] and `PETSc` [69] solvers. Finally, `NGSolve` inherently runs multi-threaded and can be run in parallel over multiple nodes if compiled with MPI support [70].

## 4.2 Code Structure

The basic structure of `OpenCmp` is shown in figure 4.1.

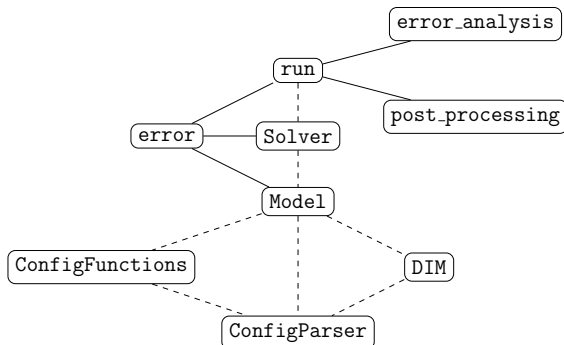


Figure 4.1: The code structure of `OpenCmp`. Solid lines indicate imported modules and dashed lines indicate initialized classes.

### 4.2.1 Basic Workflow

A typical simulation run proceeds as follows:

1. The user creates a main simulation directory to hold user-specified simulation parameters.
2. `run` is called through the command line and pointed to the main simulation directory.
3. `run` sets the run parameters.

4. `run` initializes `Solver` to control the time-stepping scheme.
5. `Solver` initializes `Model` to construct the spatial discretization.
6. `Model` initializes `ConfigFunctions` to load, store, and update the user-specified simulation parameters. It also initializes `DIM` if the diffuse interface method is being used.
7. `Solver` runs the simulation for the specified length of time, saving results to file at a user-specified frequency.
8. Once the simulation has finished, `run` may call `error` and `error_analysis` to compute error metrics. It may also call `post_processing` for processing of saved results.

The following sections explain the major components of `OpenCMP` in more detail.

### 4.2.2 `run`

The `run` module controls the execution of a simulation. It is called from the command line to begin a simulation and sets the run parameters, such as the number of threads used for multi-threading and what level of messages and warnings to display to the user. `run` then initializes `Solver` and runs the simulation. Once the simulation has finished, `run` calls `error` and `error_analysis` to compute user-specified error metrics and calls `post_processing` for conversion of saved simulation results to other file formats. This structure separates the main simulation run from the post-processing and, in the future, pre-processing. It allows stacking any arbitrary combination of individual post-processing steps. `Solver` and `Model` code is also kept independent of post-processing code for ease of development.

### 4.2.3 `Solver`

`OpenCMP` uses the method of lines [71]; partial differential equations are first discretized in space using the finite element method then treated as ordinary differential equations in time. Hence the division between the `Solver` class, which controls the time-stepping, and the `Model` class, which controls the spatial discretization.

Three main sub-classes inherit from the base solver class: the stationary solver class, the transient multi-step solver class, and the transient Runge-Kutta solver class. The two base transient solver classes reflect structural differences in the pattern of information flow for multi-step schemes and Runge-Kutta schemes. Sub-classes for various adaptive time-stepping schemes further inherit from either of these two transient solver classes. The exact inheritance structure is shown in figure 4.2. Note that Runge-Kutta methods

are more properly a subset of single-step methods [71]. `OpenCmp` naming conventions use “Runge-Kutta” simply because the only single-step methods currently implemented are Runge-Kutta schemes.

During a simulation run, `Solver` loads user-specified time-stepping parameters and initializes `Model`. It loads the initial conditions, assembles the mass and load matrices from the finite element weak form, updates the preconditioner, and then proceeds through the time integration. This includes updating boundary conditions and model parameter values, computing global error metrics, and saving results at each time step. The adaptive time-stepping classes contain additional methods to estimate local error at each time step and modify the size of the time step accordingly.

A separate module, `time_integration_schemes`, specifies the exact combination of spatial and temporal terms to produce the weak form for each time discretization scheme implemented in `OpenCmp`. Spatial terms are obtained from `Model` as bilinear and linear forms. These are evaluated at different time steps and combined with temporal terms to form the final weak form. This structure allows new time discretization schemes to be added without significantly modifying `Solver` code by simply adding them to `time_integration_schemes` and specifying whether they are multi-step or Runge-Kutta schemes.

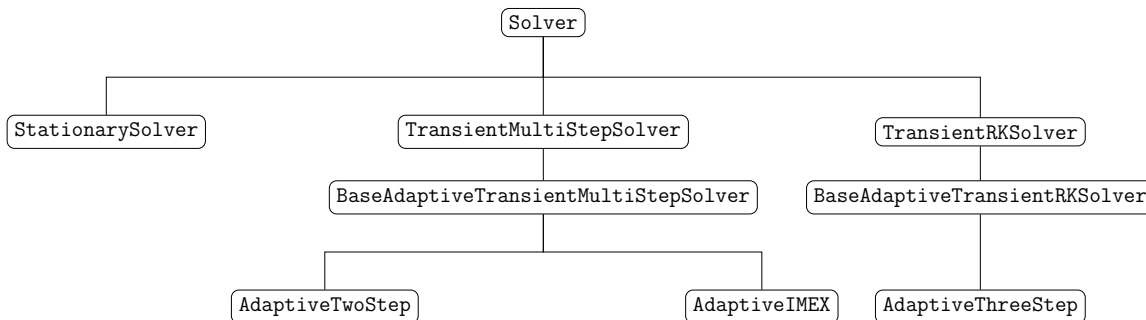


Figure 4.2: The inheritance structure of the `Solver` class.

#### 4.2.4 Model

`Model` holds the spatial discretization of the differential equations. It loads the mesh and user-specified parameter values, produces the spatial terms of the finite element weak form, and constructs the preconditioner and matrix solver for the final system of equations.

The inheritance structure, shown in figure 4.3, reflects the interrelations between many of the models currently implemented in `OpenCmp`. Two main model classes inherit from the

base model class: `Poisson` and `INS`. The remaining model classes, `MultiComponentINS` and `Stokes`, inherit from `INS`. This allows code reuse, for example, `Stokes` uses methods defined in `INS` for loading model parameters and constructing the finite element space. However, there is still separation between different models with different weak forms. Thus, a new model can be added simply by adding a new sub-class for said model without affecting the code for any existing models.

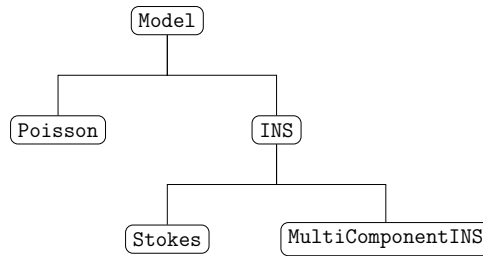


Figure 4.3: The inheritance structure of the `Model` class.

#### 4.2.5 ConfigFunctions

`ConfigFunctions` is a helper class with functionality to load, store, and update user-specified parameters. There are four main sub-classes of `ConfigFunctions` for boundary conditions, initial conditions, model parameters, and reference solutions/error metrics. The exact inheritance structure is shown in figure 4.4. All of these sub-classes are initialized by `Model` during its initialization.

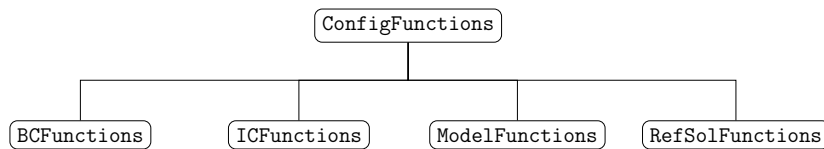


Figure 4.4: The inheritance structure of the `ConfigFunctions` class.

#### 4.2.6 DIM

`DIM` is an optional class that is initialized by `Model` if the diffuse interface method is to be used. It loads all user-specified parameters related to the creation of the phase fields, such as the interface width parameter. `DIM` is also responsible for creating the phase fields or

loading them from file and creating the masks used to apply multiple boundary conditions along the diffuse interface.

#### 4.2.7 ConfigParser

`ConfigParser` is another helper class specifically for parsing values from the configuration files used to hold user-specified parameters. It also stores the defaults used to populate model parameters when their values are not explicitly defined by the user. Various `ConfigParser` objects are initialized from each of the simulation configuration files and later called by `Model`, `ConfigFunctions`, and `DIM`.

#### 4.2.8 error\_analysis and error

The `error_analysis` module and `error` helper module handle error analysis. `error` provides functionality to compute various error metrics from a single simulation result, while `error_analysis` handles error metrics that require multiple simulation runs. At the moment, `error_analysis` is only used to run mesh or polynomial order refinement convergence tests. It includes functionality to reset the simulation to the starting time and to refine and update the mesh and finite element space for successive simulations.

#### 4.2.9 post\_processing

`post_processing` is a helper module for converting saved results to different file formats. `OpenCMP` supports saving simulation results to the native `NGSolve` [13] format, `.sol`, or to `.vtk` for visualization in `ParaView` [66]. However, the conversion from `.sol` to `.vtk` is quite time-consuming and resource-intensive. Thus, for efficiency, `post_processing` is used to convert all results to `.vtk` only after the simulation has finished. It can also be called as a separate process during the simulation run to convert results as they are obtained. Finally, `post_processing` organizes the `.vtk` filenames into a `.pvd` file so transient simulations can be visualized in time order.

### 4.3 User Interface

`OpenCMP` has a configuration file-based user interface. As discussed in section 3.1.3, a configuration file-based user interface is more suited to pre-implemented models than a

script-based interface while still being suited to automation and version control. It is also easier to develop than a graphical user interface. Details on the structure of these configuration files and the use of the command line to run `OpenCMP` are given in the following sections.

As `OpenCMP` is an open-source package, its full code base is available for modification by the user. Thus, users can treat `OpenCMP` as a script-based interface or extend the configuration file-based interface with new models, time discretization schemes, or simulation parameters. However, this will not be discussed further as it is not expected to be a common use case.

### 4.3.1 Configuration Files

For clarity and to prevent overlong configuration files, `OpenCMP` uses separate configuration files for the main simulation parameters, boundary conditions, initial conditions, model parameters, reference solutions, and diffuse interface method parameters. Thus, setting up a simulation requires creating a main simulation directory to hold all of these configuration files. Furthermore, since `OpenCMP` allows parameter values to be loaded from file instead of specified in closed form, the main simulation directory is split into multiple sub-directories to hold specific configuration files and their relevant additional files. An example simulation directory is shown in figure 4.5. In this example the diffuse interface method is used, so sub-directories and configuration files are added for diffuse interface parameters and boundary conditions. The initial condition is loaded from file and this file, `ic.sol`, is placed in the initial condition sub-directory. An output sub-directory is also shown and contains saved simulation results at two different time steps.

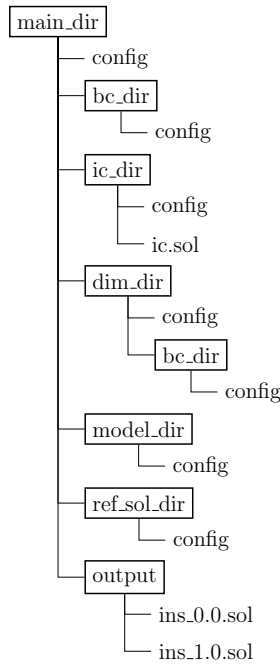


Figure 4.5: An example simulation directory structure.

Within the configuration files themselves, emphasis is placed on readability and ease of use while still allowing extensive customization. Figure 4.6 shows an example configuration file for the main simulation directory. This is a human-readable .txt file accessible to the user without translation by `OpenCmp`. It is also very amenable to version control.

`OpenCmp` uses the `configparser` library [72] to read configuration files which allows a deceptively simple syntax. Simulation parameters are organized into categories by headers, denoted by `[]`, and their values are given through the `=` marker with the `->` marker used for multi-level values. There is no need for dictionaries, as in `OpenFOAM`® [20] configuration files, or the C++ syntax used by `MOOSE` [46] configuration files. With reference documentation for the available simulation parameters it should be intuitive and easy for new users to construct their own configuration files.

Figure 4.6 is a typical example of a transient incompressible Navier-Stokes simulation, but many of the simulation parameters specified in the configuration file have default values and could have been omitted. For example, the user has specified that the conjugate gradient solver be used with a multigrid preconditioner. The user could instead have omitted both lines and `OpenCmp` would default to a direct solver and preconditioner. In this case, `OpenCmp` would warn the user of the default values, in case the omission was



unintentional. There are also many additional options not shown in Figure 4.6 but available if desired. For example, the convergence criteria of the solver—its residual error tolerance and the maximum number of iterations—could be specified. This design avoids excessively large configuration files that overwhelm the user with options while still offering experienced users a high level of control.

```
[MESH]
filename = mesh_files/channel.vol

[FINITE ELEMENT SPACE]
elements = u -> HDiv
          p -> L2
interpolant_order = 3

[DG]
DG = True
interior_penalty_coefficient = 10.0

[SOLVER]
solver = CG
preconditioner = multigrid
linearization_method = Oseen
nonlinear_tolerance = relative -> 1e-6
                    absolute -> 1e-8
nonlinear_max_iterations = 5

[TRANSIENT]
transient = True
scheme = crank nicolson
time_range = 0.0, 10
dt = 5e-4

[VISUALIZATION]
save_to_file = True
save_type = .vtu
save_frequency = 0.1, time

[ERROR ANALYSIS]
check_error = True

[OTHER]
num_threads = 6
model = INS
run_dir = ins_example
```

Figure 4.6: An example configuration file for the main simulation directory.

Figure 4.7 shows an example configuration file for the boundary condition sub-directory. Headers group the different types of boundary condition. Then, each boundary condition is specified as the model variable it acts upon, the mesh marker where it is applied, and the value it takes. For example, the second line in figure 4.7 specifies a Dirichlet boundary

condition of  $u = -\sin(\pi x)\sin(\pi t)$  applied at the top boundary (in this example of a unit square).

Some computational multiphysics software packages implement many very specific forms of each boundary condition. For example, `OpenFOAM`<sup>®</sup> provides the `zeroGradient` boundary condition specifically for insulated boundaries separate from the more general `fixedGradient` boundary condition [73]. However, this becomes needlessly complicated for the user and adds redundant code to the software package. `OpenCMP` differentiates boundary conditions by their mathematical appearance in the finite element weak form but otherwise leaves them highly general. In the case of figure 4.7, insulated boundary conditions and nonzero flux boundary conditions would both fall under the general Neumann boundary condition header.

`OpenCMP` is also unique in allowing a symbolic depiction of mathematical expressions within its configuration files through the use of the `pyarsing` library [74]. The second and the last lines of text in figure 4.7 are both easily recognizable as spatially- and temporally-varying sinusoidal functions. The syntax is intuitive, readable, and short. Using `SU2`, for example, this single-line depiction of each boundary condition would not be possible, instead the user would need to specify the values at each boundary mesh node in a separate file and point the configuration file to said file [75]. However, to be comprehensive, `OpenCMP` can still load parameter values from file. For example, the third line in figure 4.7 specifies that the Dirichlet boundary condition for the bottom boundary will be loaded from the file “bottom\_bc.sol”. This is particularly useful for initial conditions; for example, it is very common for incompressible Navier-Stokes simulations to be initialized by a Stokes simulation on the same domain. `OpenCMP` can also obtain parameter values from functions imported from a separate `Python` [48] script (see the second last line of figure 4.7, which imports the function `right_bc`). This gives the user access to mathematical operators not yet included in `ConfigParser` as well as external `Python` libraries.

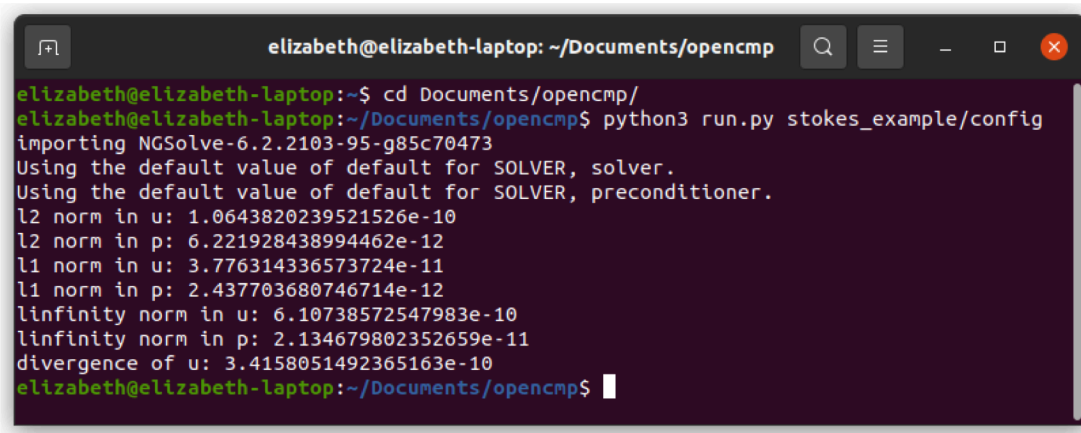
```
[DIRICHLET]
u = top    -> -sin(pi * x) * sin(pi * t)
    bottom -> bottom_bc.sol

[NEUMANN]
u = right  -> IMPORT(right_bc)
    left   -> -pi * cos(pi * y) * sin(pi * t)
```

Figure 4.7: An example configuration file for the boundary condition sub-directory.

### 4.3.2 Command Line Interaction

Once a simulation directory and configuration files have been generated, `OpenCMP` is run through the command line. An example is shown in Figure 4.8; the “`stokes_example`” directory has been set up to specify the simulation parameters and calling the `run` module pointing to the main configuration file of said directory runs the simulation. Messages appear to warn the user that default values will be used for some of the important simulation parameters, in this case the solver and preconditioner. Then, once the simulation has finished, the user-specified error metrics are computed and shown. In the case of a transient simulation, time step information would also be output over the course of the simulation to track its progress.

A terminal window titled "elizabeth@elizabeth-laptop: ~/Documents/opencmp" showing the execution of a Python script. The output displays various error metrics for a simulation. The terminal text is as follows:

```
elizabeth@elizabeth-laptop:~$ cd Documents/opencmp/  
elizabeth@elizabeth-laptop:~/Documents/opencmp$ python3 run.py stokes_example/config  
importing NGSolve-6.2.2103-95-g85c70473  
Using the default value of default for SOLVER, solver.  
Using the default value of default for SOLVER, preconditioner.  
l2 norm in u: 1.0643820239521526e-10  
l2 norm in p: 6.221928438994462e-12  
l1 norm in u: 3.776314336573724e-11  
l1 norm in p: 2.437703680746714e-12  
linfinity norm in u: 6.10738572547983e-10  
linfinity norm in p: 2.134679802352659e-11  
divergence of u: 3.4158051492365163e-10  
elizabeth@elizabeth-laptop:~/Documents/opencmp$
```

Figure 4.8: Command line output of an example `OpenCMP` simulation.

## 4.4 Capabilities

Table 4.1 summarizes the current capabilities of `OpenCMP`. `OpenCMP` covers the bulk of the typical simulation workflow as described in section 3.1.2. It primarily lacks geometry creation and meshing capabilities. However, these are available through `Netgen` [13], so could in the future be integrated into the `OpenCMP` user interface. `OpenCMP` is also currently reliant on external use of `Netgen` (or `ParaView` [66]) for visualization of saved results. It would be desirable to give users the option of opening `Netgen` for visualization through the `OpenCMP` user interface. `OpenCMP` currently offers many of the fluid flow models listed in section 3.1.1; it primarily lacks multi-phase and turbulent flow. To be fully applicable to

most chemical engineering applications, heat transfer models must also be added. `OpenCmp` provides a good selection of numerical methods, focusing on the finite element method for increased simulation accuracy and offering discontinuous Galerkin implementations for improved simulation stability. Many time discretization schemes are provided, including higher order ones to increase simulation accuracy. Adaptive time-stepping is also available for improved computational efficiency. The available error metrics are quite comprehensive and include automated convergence testing. However, the only design parameter which can be computed during post-processing is surface traction. In the future, it would be preferable for users to be able to specify any design parameter or general mathematical expression to be included in the post-processing. `OpenCmp` contains a full implementation of the diffuse interface method as described in work by Monte *et al* [8]. It offers simulations with a stationary phase field or rigid body motion of the phase field, as will be discussed further in Chapter 6. Performance-wise `OpenCmp` can be run multi-threaded with a user-specified number of threads. In the future, it would also be desirable to run `OpenCmp` fully parallelized over multiple nodes with MPI [70].

Table 4.1: Current capabilities of `OpenCmp`.

Meshing	Accepts <code>Netgen</code> [13] or <code>Gmsh</code> [64] meshes
Numerical Methods	Standard finite element method Discontinuous Galerkin finite element method
Models	Poisson equation Stokes equations Incompressible Navier-Stokes equations Multi-component flow Reacting flow
Time Discretization Schemes	First-, second-, or third-order schemes Adaptive time-stepping
Solvers	Direct or iterative solvers Direct, Jacobi, or multigrid preconditioners Oseen or IMEX linearization of nonlinear models
Post-Processing	Error norms calculated based on reference solutions Divergence of velocity field Magnitude of discontinuities in solution field Surface traction calculated at specified boundaries Mesh and polynomial refinement convergence tests Saves results to <code>Netgen</code> [13] or <code>ParaView</code> [66] format
Diffuse Interface Method	Automatic phase field and mesh generation from CAD Rigid body motion
Performance	Multi-threading

# Chapter 5

## Performance Verification

This chapter describes the verification of the numerical implementation of `OpenCMP`. Problems with known analytical solutions are used to confirm that the various models implemented in `OpenCMP` show expected error convergence behaviour, including the expected convergence rates for mesh refinement and time step refinement. Some common benchmarks are also shown to confirm that `OpenCMP` meets the performance expected of computational multiphysics packages. The fidelity of the models to real-world processes is not assessed as proper model selection will be the responsibility of the users. General software quality, reliability, repeatability, and coverage is also not discussed here. These are tested by a suite of unit tests which are publicly available as part of the `OpenCMP` package.

In most cases in the following sections, error is examined in the L2 norm:

$$\text{error}_2 = \sqrt{\int_{\Omega} (u - u_{exact})^2 dx} \quad (5.1)$$

where  $u$  is the model variable or computed parameter of interest and  $u_{exact}$  is a known reference solution. Error may also be examined in the L1 norm:

$$\text{error}_1 = \int_{\Omega} |u - u_{exact}| dx \quad (5.2)$$

or the  $L_{\infty}$  norm:

$$\text{error}_{\infty} = \max(|u - u_{exact}|) \quad (5.3)$$

In the case of incompressible flow, the divergence of the velocity field is computed to ensure it is close to zero:

$$\nabla \cdot \mathbf{u} = \sqrt{\int_{\Omega} (\nabla \cdot \mathbf{u})^2 dx} \quad (5.4)$$

When the discontinuous Galerkin method is used the degree of discontinuity in the final solution field is also of interest:

$$[[u]] = \sqrt{\sum_{\mathcal{K} \in \Omega} \int_{\mathcal{F}} (u^+ - u^-)^2 ds} \quad (5.5)$$

Many of the following verification tests include mesh and time step refinement to determine error convergence rates in the L2 norm. When the standard finite element method is used, mesh refinement convergence is expected to be of order  $n$  where  $n$  is the order of the polynomial interpolant. The discontinuous Galerkin method is expected to give convergence at the order of  $n + 1$  [11]. Time step convergence rates are expected to agree with the order of the time discretization scheme used. For example, a first-order scheme like implicit Euler is expected to give first-order convergence [71].

The online resource by Arnold and Logg [76] gives a visual overview of typical finite element spaces. This work uses the H1 space discretized by Lagrange elements as the standard for scalar variables. For incompressible flow problems, the Taylor-Hood finite element pair—VectorH1 for velocity and H1 for pressure—is standard. Note that when using this finite element pair, the polynomial interpolant order of the velocity space must be one higher than that of the pressure space to satisfy the inf-sup condition [31]. If the discontinuous Galerkin method is used for incompressible flow problems it is preferable to use HDiv for velocity and L2 for pressure. As discussed in section 2.2.2, this finite element pair strongly enforces the incompressibility constraint without requiring additions to the finite element weak form. The same constraint on polynomial interpolant order is needed to satisfy the inf-sup condition as with the Taylor-Hood element pair [11].

## 5.1 The Poisson Equation

### 5.1.1 The Steady-State Poisson Equation

The problem, which includes a nonzero source term and mixed Dirichlet and Neumann boundary conditions, is as follows:

$$-\nabla^2 u = 2\pi^2 \sin(\pi x) \cos(\pi y) \quad \text{in } x \in [0, 1], y \in [0, 1] \quad (5.6)$$

$$u = \pm \sin(\pi x) \quad \text{on } y = 0, 1 \quad (5.7)$$

$$-\mathbf{n} \cdot \nabla u = -\pi \cos(\pi y) \quad \text{on } x = 0, 1 \quad (5.8)$$

which has the exact solution:

$$u = \sin(\pi x) \cos(\pi y) \quad (5.9)$$

The standard H1 finite element space is used with a polynomial order of three. A mesh convergence test is conducted beginning with a very coarse mesh and, as shown in figure 5.1, a super-optimal convergence rate of one more than the polynomial order is achieved.

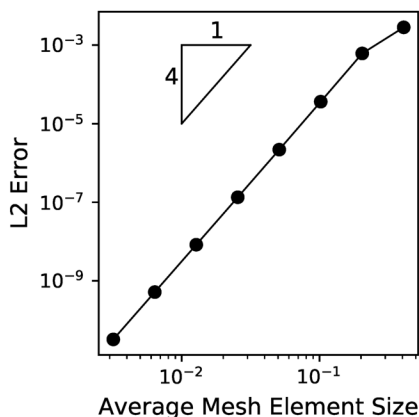


Figure 5.1: Mesh convergence for the Poisson equation.

The simulation output on the finest mesh is visualized in figure 5.2 and appears very similar to the known exact solution as expected given the low simulation error.



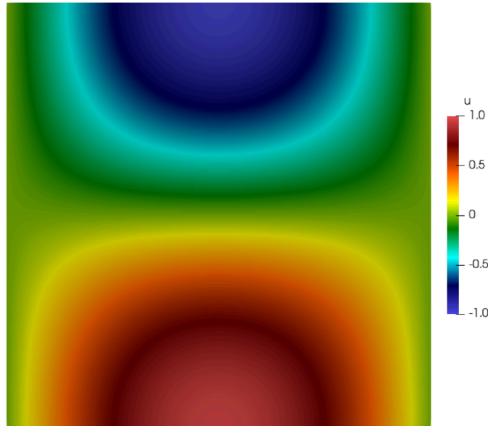


Figure 5.2: Simulation results for the Poisson equation.

### 5.1.2 The Transient Poisson Equation

The transient Poisson equation is also tested. The problem is similar to that used in section 5.1.1 but with a sinusoidal variation in the magnitude of the source term.

$$\frac{\partial u}{\partial t} - \nabla^2 u = f \quad \text{in } x \in [0, 1], y \in [0, 1] \quad (5.10)$$

$$u(t = 0) = 0 \quad \text{in } x \in [0, 1], y \in [0, 1] \quad (5.11)$$

$$u = \pm \sin(\pi x) \sin(\pi t) \quad \text{on } y = 0, 1 \quad (5.12)$$

$$-\mathbf{n} \cdot \nabla u = -\pi \cos(\pi y) \sin(\pi t) \quad \text{on } x = 0, 1 \quad (5.13)$$

with:

$$f = 2\pi^2 \sin(\pi x) \cos(\pi y) \sin(\pi t) + \pi \sin(\pi x) \cos(\pi y) \cos(\pi t) \quad (5.14)$$

which has the exact solution:

$$u = \sin(\pi x) \cos(\pi y) \sin(\pi t) \quad (5.15)$$

Again an H1 finite element space is used, this time with a polynomial order of two. A mesh convergence test is conducted, comparing the error after 1 s for a series of increasingly refined meshes. The time step is held constant at  $\Delta t = 1 \times 10^{-4}$  s to ensure the error in the spatial discretization outweighs the error in the temporal discretization. As shown in figure 5.3a, a super-optimal convergence rate is achieved again. Next a time step convergence

test is conducted, starting with a time step of 0.2s. A refined mesh with 938 elements is used to ensure error is dominated by the temporal discretization error. Implicit Euler is used as the time discretization scheme and, as shown in figure 5.3b, the convergence is first-order as expected.

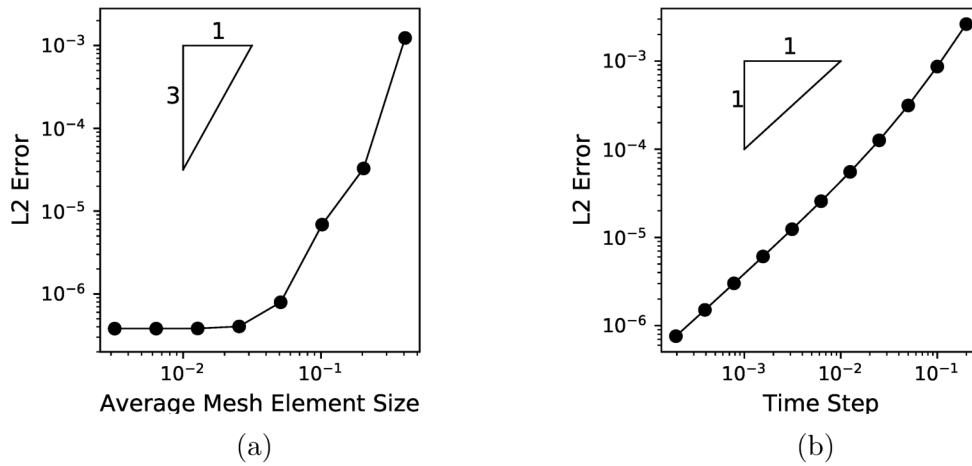


Figure 5.3: (a) mesh convergence and (b) time step convergence for the transient Poisson equation.

Figure 5.4 visualizes the simulation output at several different times. As expected, they appear similar to figure 5.2 but with a time-varying magnitude.

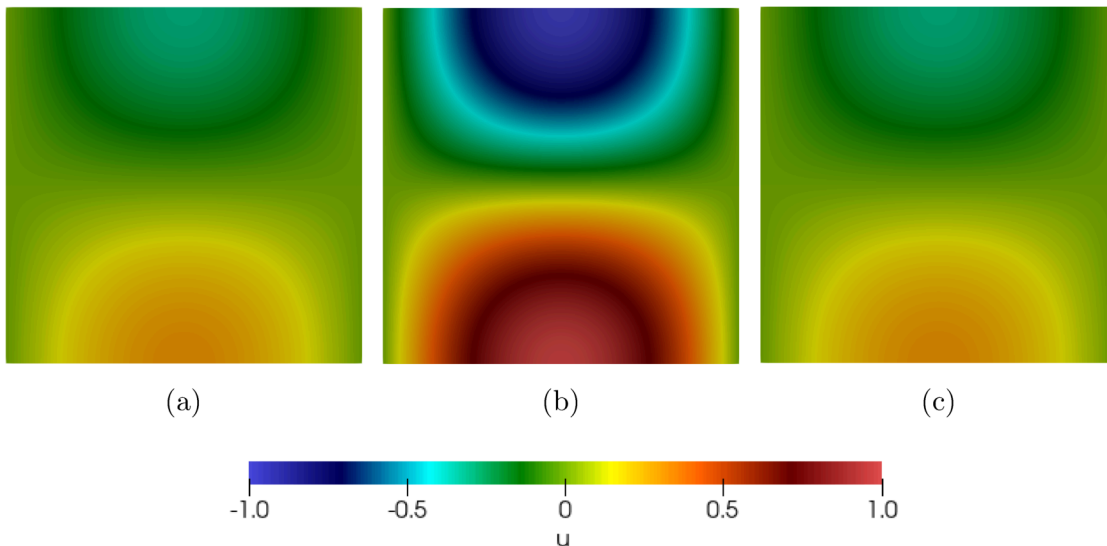


Figure 5.4: Simulation results at (a)  $t = 0.1$  s, (b)  $t = 0.5$  s, and (c)  $t = 0.9$  s for the transient Poisson equation.

## 5.2 The Stokes Equations

### 5.2.1 Poiseuille Flow

One common example of Stokes flow with a known analytical solution is Poiseuille flow, channel flow driven by an imposed pressure gradient:

$$-\nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{0} \quad \text{in } x \in [0, 1], y \in [0, 0.2] \quad (5.16)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } x \in [0, 1], y \in [0, 0.2] \quad (5.17)$$

$$p = 10 \quad \text{on } x = 0 \quad (5.18)$$

$$p = 5 \quad \text{on } x = 1 \quad (5.19)$$

The kinematic viscosity is chosen to be  $\nu = 0.01$ , giving a Reynolds number of 50 which is firmly in the laminar regime. The exact velocity and pressure fields are:

$$\mathbf{u} = 250 (0.2y - y^2) \boldsymbol{\delta}_x \quad (5.20)$$

$$p = 5(1 - x) + 5 \quad (5.21)$$

OpenCMP does not allow pressure Dirichlet boundary conditions to be directly specified. As noted by Rempfer [40], it is challenging to specify pressure boundary conditions for incompressible flow which are both mathematically well-posed and consistent with the governing equations (the Stokes or incompressible Navier-Stokes equations). However, in this case, since the flow is unidirectional and perpendicular to the inlet and outlet, the pressure boundary conditions can be imposed through normal stress boundary conditions:

$$\mathbf{n} \cdot (-\nu \nabla \mathbf{u} + p \mathbb{I}) = h \quad \text{on } \Gamma \quad (5.22)$$

$$0 + \mathbf{n} \cdot p \mathbb{I} = h \quad \text{on } \Gamma \quad (5.23)$$

$$p = h \quad \text{on } \Gamma \quad (5.24)$$

The discontinuous Galerkin method is used with the HDiv-L2 mixed finite element space in order to strongly enforce the incompressibility constraint. A polynomial order of three is used for velocity and two for pressure to satisfy the inf-sup condition. Error is mesh-independent for any reasonable mesh, so results are given (see table 5.1) for one single mesh with 128 elements. All error norms are quite low, indicating an accurate result. The incompressibility constraint is satisfied, as shown by a velocity divergence close to zero. The discontinuities in the velocity and pressure fields are also very low as expected; this problem has a continuous solution.

Table 5.1: Error results for Poiseuille flow.

Error Metric	Value
$\mathbf{u}$ L2 Norm	$1.064 \times 10^{-10}$
$p$ L2 Norm	$6.219 \times 10^{-11}$
$\mathbf{u}$ L1 Norm	$3.776 \times 10^{-11}$
$p$ L1 Norm	$2.437 \times 10^{-11}$
$\mathbf{u}$ L $\infty$ Norm	$2.824 \times 10^{-11}$
$p$ L $\infty$ Norm	$2.079 \times 10^{-10}$
$\nabla \cdot \mathbf{u}$	$3.416 \times 10^{-10}$
$[[\mathbf{u}]]$	$6.694 \times 10^{-14}$
$[[p]]$	$4.112 \times 10^{-12}$

The velocity and pressure fields are visualized in figure 5.5 and appear very similar to the exact solution.

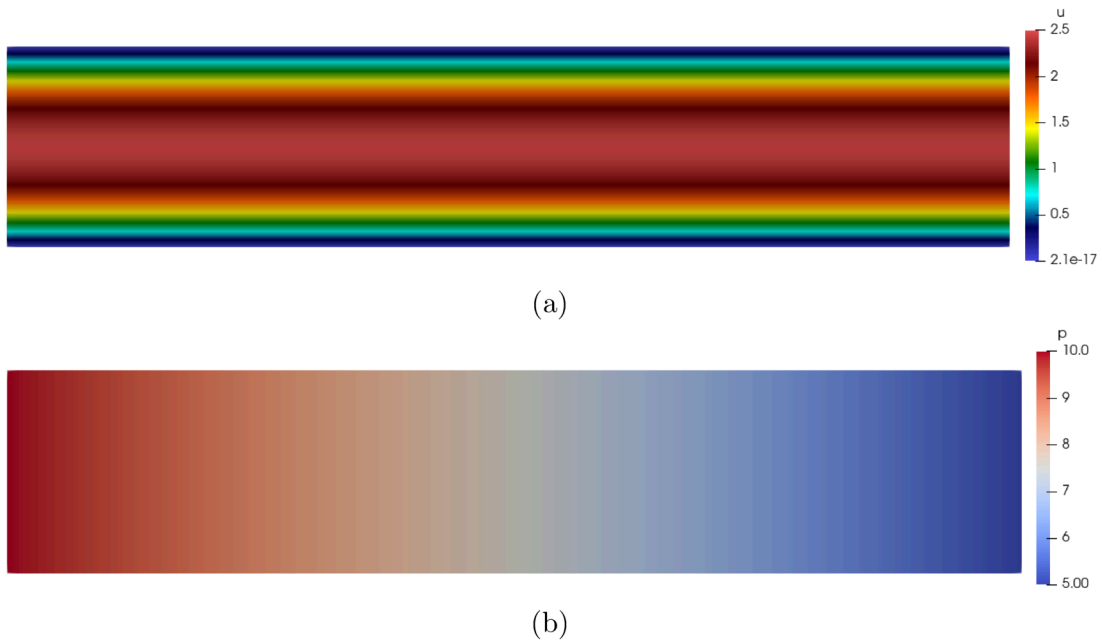


Figure 5.5: Simulation results of (a) velocity and (b) pressure for Poiseuille flow.

### 5.2.2 Couette Flow

A second common example of Stokes flow is Couette flow, channel flow drive by the movement of one wall of the channel:

$$-\nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{0} \quad \text{in } x \in [0, 1], y \in [0, 0.2] \quad (5.25)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } x \in [0, 1], y \in [0, 0.2] \quad (5.26)$$

$$\mathbf{u} = \mathbf{0} \quad \text{on } y = 0 \quad (5.27)$$

$$\mathbf{u} = \delta_x \quad \text{on } y = 1 \quad (5.28)$$

The kinematic viscosity is chosen to be  $\nu = 0.01$ , giving a Reynolds number of 20 which is firmly in the laminar regime. The exact velocity and pressure fields are:

$$\mathbf{u} = 5y\delta_x \quad (5.29)$$

$$p = 0 \quad (5.30)$$

This discontinuous Galerkin method is used with the HDiv-L2 mixed finite element space in order to strongly enforce the incompressibility constraint. A polynomial order of

three is used for velocity and two for pressure to satisfy the inf-sup condition. Error is again mesh-independent for any reasonable mesh, so results are given (see table 5.2) for one single mesh with 128 elements. All error norms are quite low, indicating an accurate result. The incompressibility constraint is satisfied, as shown by a velocity divergence close to zero. The discontinuities in the velocity and pressure fields are also very low as expected; this problem has a continuous solution.

Table 5.2: Error results for Couette flow.

Error Metric	Value
$\mathbf{u}$ L2 Norm	$2.619 \times 10^{-14}$
$p$ L2 Norm	$3.062 \times 10^{-14}$
$\mathbf{u}$ L1 Norm	$9.573 \times 10^{-15}$
$p$ L1 Norm	$9.595 \times 10^{-15}$
$\mathbf{u}$ L $\infty$ Norm	$2.818 \times 10^{-14}$
$p$ L $\infty$ Norm	$3.508 \times 10^{-13}$
$\nabla \cdot \mathbf{u}$	$6.837 \times 10^{-16}$
$[[\mathbf{u}]]$	$1.894 \times 10^{-14}$
$[[p]]$	$7.024 \times 10^{-13}$

The velocity and pressure fields are visualized in figure 5.6 and appear very similar to the exact solution. There are minor discontinuities in the pressure field, but they are simply numerical error. The pressure field is effectively uniformly zero as expected.

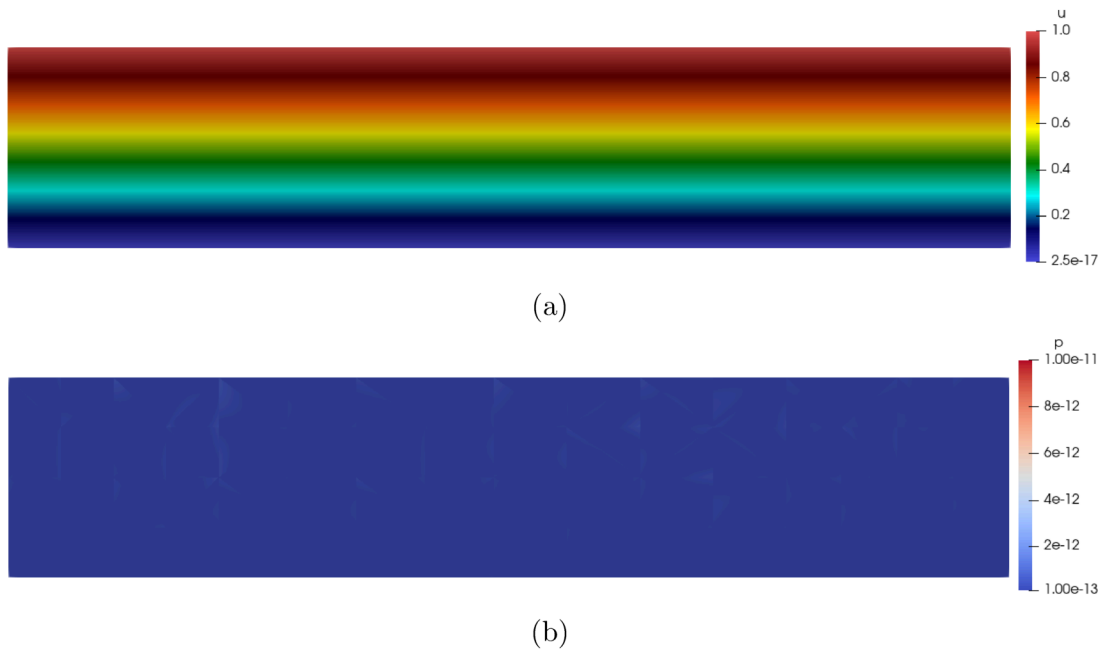


Figure 5.6: Simulation results of (a) velocity and (b) pressure for Couette flow.

### 5.2.3 The Transient Stokes Equations

The transient Stokes equations are also tested using a problem constructed to have an analytical solution:

$$\frac{\partial \mathbf{u}}{\partial t} - \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } x \in [0, \pi], y \in [0, \pi] \quad (5.31)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } x \in [0, \pi], y \in [0, \pi] \quad (5.32)$$

$$\mathbf{u}(t=0) = \sin(x) \sin(y) \boldsymbol{\delta}_x + \cos(x) \cos(y) \boldsymbol{\delta}_y \quad \text{in } x \in [0, \pi], y \in [0, \pi] \quad (5.33)$$

$$p(t=0) = \sin(x) \cos(y) \quad \text{in } x \in [0, \pi], y \in [0, \pi] \quad (5.34)$$

$$\mathbf{u} = \pm \cos(y) e^{-t} \boldsymbol{\delta}_y \quad \text{on } x = 0, \pi \quad (5.35)$$

$$\mathbf{u} = \pm \cos(x) e^{-t} \boldsymbol{\delta}_y \quad \text{on } y = 0, \pi \quad (5.36)$$

with:

$$\mathbf{f} = (\sin(x) \sin(y) + \cos(x) \cos(y)) e^{-t} \boldsymbol{\delta}_x - (\sin(x) \sin(y) - \cos(x) \cos(y)) e^{-t} \boldsymbol{\delta}_y \quad (5.37)$$

which has the exact solution:

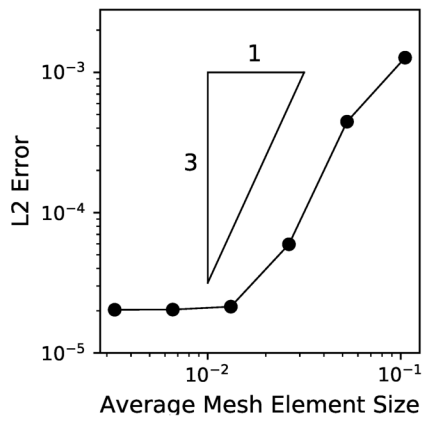
$$\mathbf{u} = \sin(x) \sin(y) e^{-t} \boldsymbol{\delta}_x + \cos(x) \cos(y) e^{-t} \boldsymbol{\delta}_y \quad (5.38)$$

$$p = \sin(x) \cos(y) e^{-t} \quad (5.39)$$

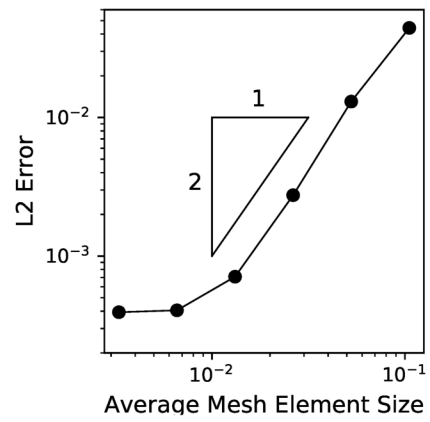
The discontinuous Galerkin method is used with the HDiv-L2 mixed finite element space in order to strongly enforce the incompressibility constraint. A polynomial order of two is used for velocity and one for pressure to satisfy the inf-sup condition. A mesh convergence test is conducted, comparing the error after 1 s for a series of increasingly refined meshes. The time step is held constant at  $\Delta t = 1 \times 10^{-3}$  s to ensure the error in the spatial discretization outweighs the error in the temporal discretization. As shown in figure 5.7a–b, the expected convergence rate of one more than the polynomial order is achieved for both velocity and pressure. Generally the convergence of the divergence of velocity would also be considered. However, due to the use of HDiv-L2 finite elements, the divergence of velocity is close to machine precision for any mesh size. Next a time step convergence test is conducted, starting with a time step of 0.2 s. A refined mesh with 2296 elements is used to ensure error is dominated by the temporal discretization error. Implicit Euler is used as the time discretization scheme and, as shown in figure 5.7c–d, both velocity and pressure have the expected first-order convergence. Again, the divergence of velocity is close to machine precision for any time step so is not considered.

Figure 5.8 and figure 5.9 visualize the velocity and pressure fields at several different times. As expected, they appear similar to the known exact solution.

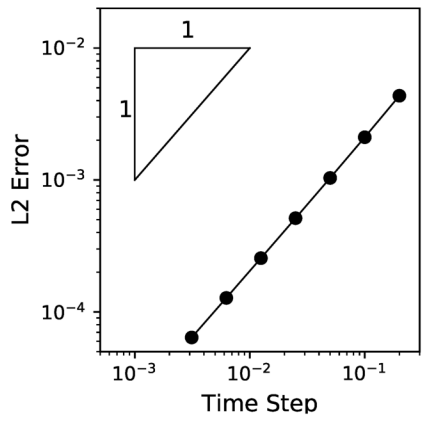




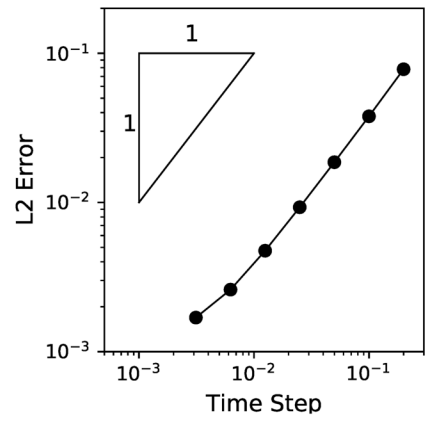
(a)



(b)



(c)



(d)

Figure 5.7: Mesh convergence for (a) velocity and (b) pressure and time step convergence for (c) velocity and (d) pressure for transient Stokes flow.

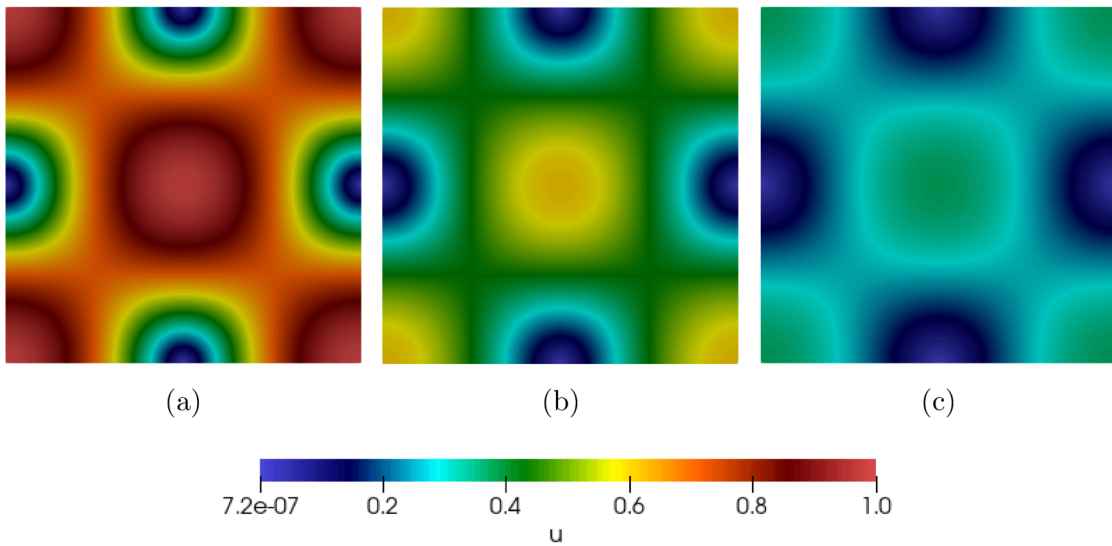


Figure 5.8: Simulation results for velocity at (a)  $t = 0$  s, (b)  $t = 0.5$  s, and (c)  $t = 1$  s for transient Stokes flow.

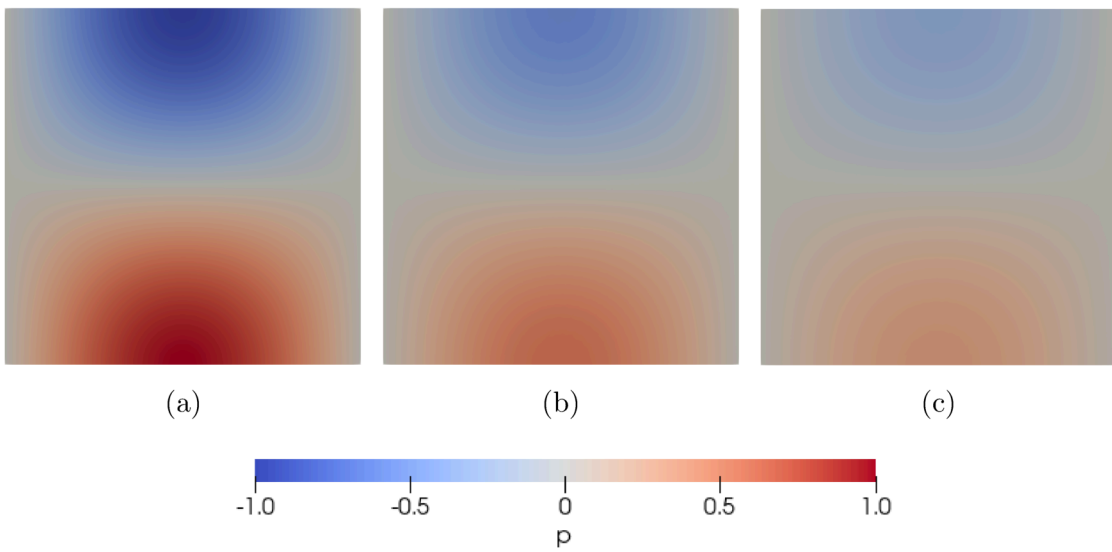


Figure 5.9: Simulation results for pressure at (a)  $t = 0$  s, (b)  $t = 0.5$  s, and (c)  $t = 1$  s for transient Stokes flow.

## 5.3 The Incompressible Navier-Stokes Equations

### 5.3.1 Taylor-Green Vortices

One common benchmark for the incompressible Navier-Stokes equations with a known analytical solution is the Taylor-Green vortices, which describes the unsteady flow of vortices as they decay to zero [77]. In two dimensions with a kinematic viscosity of one the equations are as follows:

$$\mathbf{0} = \frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) - \nabla^2 \mathbf{u} + \nabla p \quad \text{in } x \in [0, 2\pi], y \in [0, 2\pi] \quad (5.40)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } x \in [0, 2\pi], y \in [0, 2\pi] \quad (5.41)$$

$$\mathbf{u}(t=0) = -\cos(x)\sin(y)\boldsymbol{\delta}_x + \sin(x)\cos(y)\boldsymbol{\delta}_y \quad \text{in } x \in [0, 2\pi], y \in [0, 2\pi] \quad (5.42)$$

$$p(t=0) = -\frac{1}{4}(\cos(2x) + \cos(2y)) \quad \text{in } x \in [0, 2\pi], y \in [0, 2\pi] \quad (5.43)$$

$$\mathbf{u} = -\sin(y)e^{-2t}\boldsymbol{\delta}_x \quad \text{on } x = 0, 2\pi \quad (5.44)$$

$$\mathbf{u} = \sin(x)e^{-2t}\boldsymbol{\delta}_y \quad \text{on } y = 0, 2\pi \quad (5.45)$$

which has the exact solution:

$$\mathbf{u} = -\cos(x)\sin(y)e^{-2t}\boldsymbol{\delta}_x + \sin(x)\cos(y)e^{-2t}\boldsymbol{\delta}_y \quad (5.46)$$

$$p = -\frac{1}{4}(\cos(2x) + \cos(2y))e^{-4t} \quad (5.47)$$

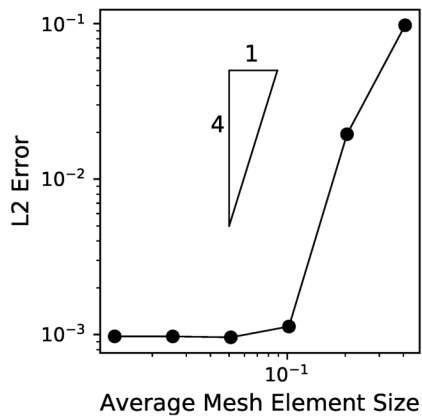
The convection term in eqn. (5.40) is nonlinear due to the dyadic product of velocity with itself. `OpenCMP` has two options for linearizing systems of nonlinear equations so they can be solved with linear solvers—Oseen-style linearization or IMEX time discretization—implementation details for which can be found in Appendix A. For this test, Oseen-style linearization is used and one of the unknown velocities is replaced by a known velocity field  $\mathbf{w}$ :

$$\nabla \cdot (\mathbf{u}\mathbf{u}) \rightarrow \nabla \cdot (\mathbf{u}\mathbf{w}) \quad (5.48)$$

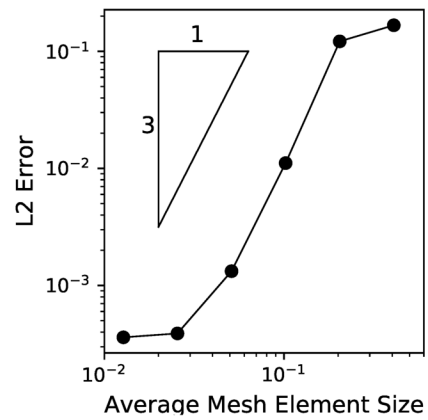
The value of the known velocity field is initially taken to be the solution of the previous time step then refined through Picard iterations [71]. The value is considered sufficiently converged at a relative tolerance of  $1 \times 10^{-4}$  and an absolute tolerance of  $1 \times 10^{-6}$  or after five iterations.

The discontinuous Galerkin method is used with the HDiv-L2 mixed finite element space in order to strongly enforce the incompressibility constraint. A polynomial order of three is used for velocity and two for pressure to satisfy the inf-sup condition. A mesh convergence test is conducted, comparing the error after 1 s for a series of increasingly refined meshes. The time step is held constant at  $\Delta t = 1 \times 10^{-3}$  s to ensure the error in the spatial discretization outweighs the error in the temporal discretization. As shown in figure 5.10a–b, the expected convergence rate of one more than the polynomial order is achieved for both velocity and pressure. Generally the convergence of the divergence of velocity would also be considered. However, due to the use of HDiv-L2 finite elements, the divergence of velocity is close to machine precision for any mesh size. Next a time step convergence test is conducted, starting with a time step of 0.2 s. A refined mesh with 378 elements is used to ensure error is dominated by the temporal discretization error. Implicit Euler is used as the time discretization scheme and, as shown in figure 5.10c–d, both velocity and pressure have the expected first-order convergence. Again, the divergence of velocity is close to machine precision for any time step so is not considered.

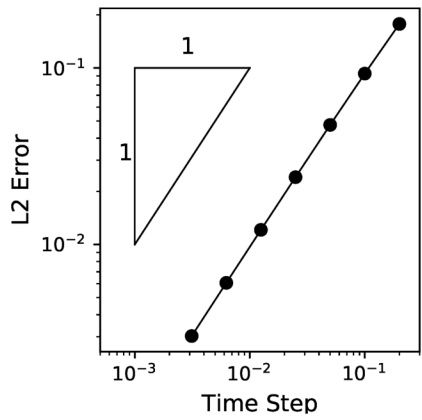
Figure 5.11 and figure 5.12 visualize the velocity and pressure fields at several different times. As expected, they appear similar to the known exact solution.



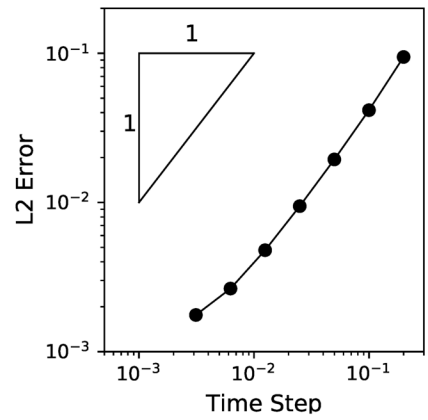
(a)



(b)



(c)



(d)

Figure 5.10: Mesh convergence for (a) velocity and (b) pressure and time step convergence for (c) velocity and (d) pressure for Taylor-Green vortices.

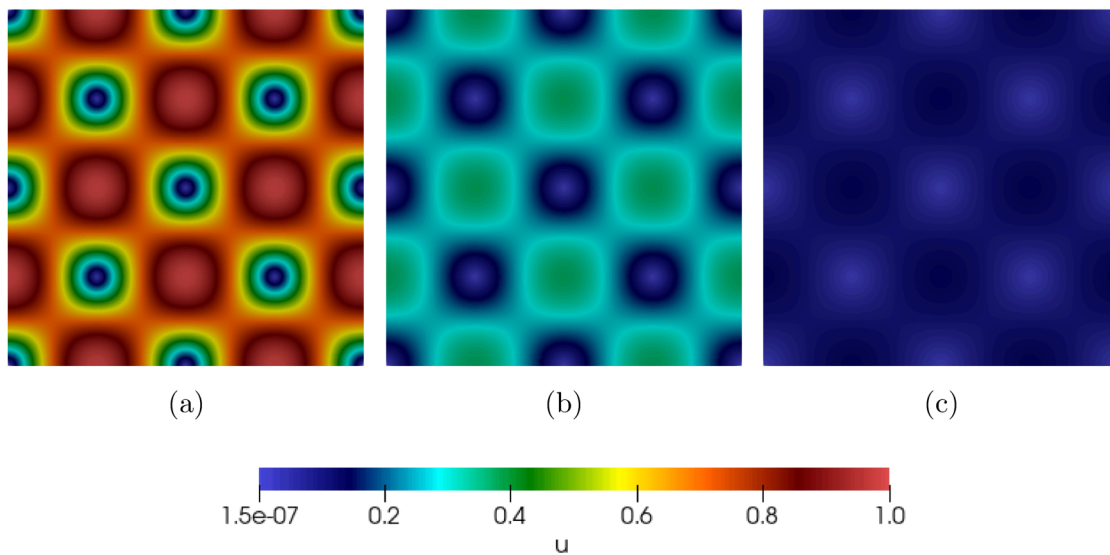


Figure 5.11: Simulation results for velocity at (a)  $t = 0$  s, (b)  $t = 0.5$  s, and (c)  $t = 1$  s for Taylor-Green vortices.

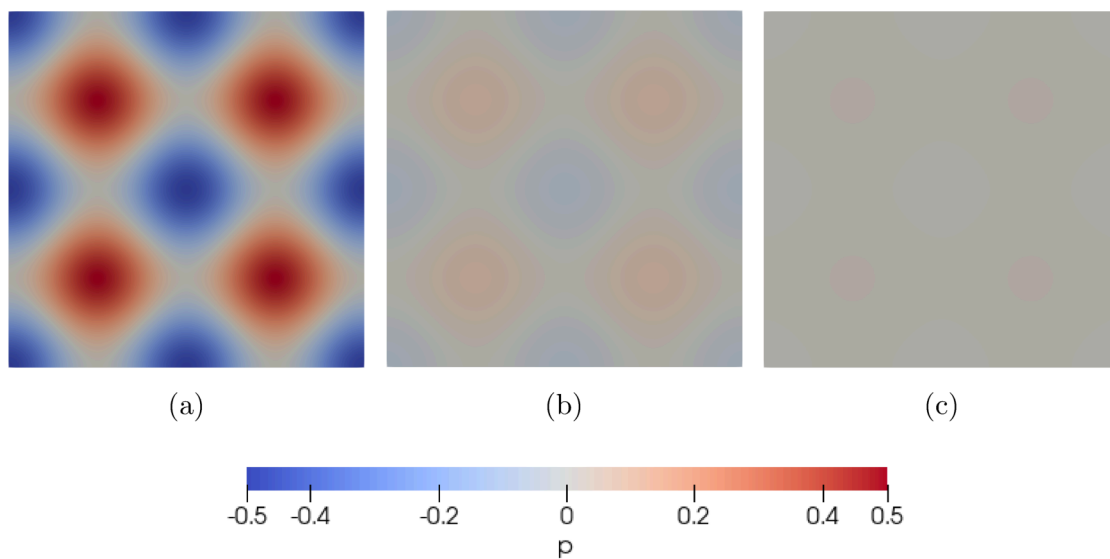


Figure 5.12: Simulation results for pressure at (a)  $t = 0$  s, (b)  $t = 0.5$  s, and (c)  $t = 1$  s for Taylor-Green vortices.

### 5.3.2 Schäfer-Turek Benchmark

Another common benchmark is the Schäfer-Turek benchmark; transient laminar flow past an immersed object which exhibits vortex-shedding [78]. The problem domain is shown in figure 5.13.

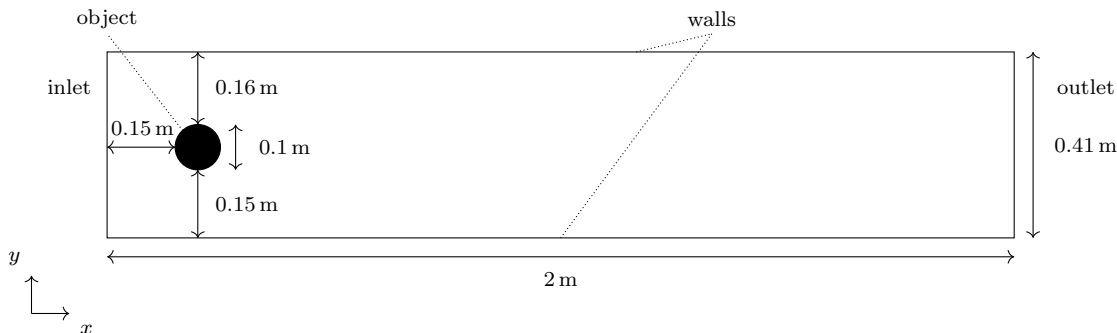


Figure 5.13: The simulation geometry for the two-dimensional Schäfer-Turek benchmark.

There are several different benchmarks defined on this domain, one of which is unsteady flow with a time-constant inlet velocity (test case 2D-2 [78]):

$$\mathbf{0} = \frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) - \nu \nabla^2 \mathbf{u} + \nabla p \quad \text{in } \Omega \quad (5.49)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \quad (5.50)$$

$$\mathbf{u} = \mathbf{0} \quad \text{on walls and object} \quad (5.51)$$

$$\mathbf{u} = \frac{6y(0.41 - y)}{0.41^2} \delta_x \quad \text{at inlet} \quad (5.52)$$

$$\mathbf{0} = \mathbf{n} \cdot (\mathbf{u}\mathbf{u} - \nu \nabla \mathbf{u} + p\mathbb{I}) - \max(\mathbf{u} \cdot \mathbf{n}, 0) \mathbf{u} \quad \text{at outlet} \quad (5.53)$$

A kinematic viscosity of  $\nu = 1 \times 10^{-3}$  results in a Reynolds number of 100. The simulation is initialized by a steady-state Stokes solve with the same boundary conditions.

Again the convection term must be linearized, in this case through the use of an IMEX scheme which treats the convection term explicitly and all remaining terms implicitly. The discontinuous Galerkin method is used with the HDiv-L2 mixed finite element space in order to strongly enforce the incompressibility constraint. A polynomial order of three is used for velocity and two for pressure to satisfy the inf-sup condition. A refined mesh with 776 elements is used and the time discretization scheme is a first-order IMEX scheme with

a time step  $\Delta t = 5 \times 10^{-4}$  s. The quantities of interest are the drag and lift coefficients, which are plotted in figure 5.14a–b. The computed maximum values, 3.2400 and 1.0227 respectively, agree reasonably well with the literature values of 3.2200–3.2400 and 0.9900–1.0100 [78].

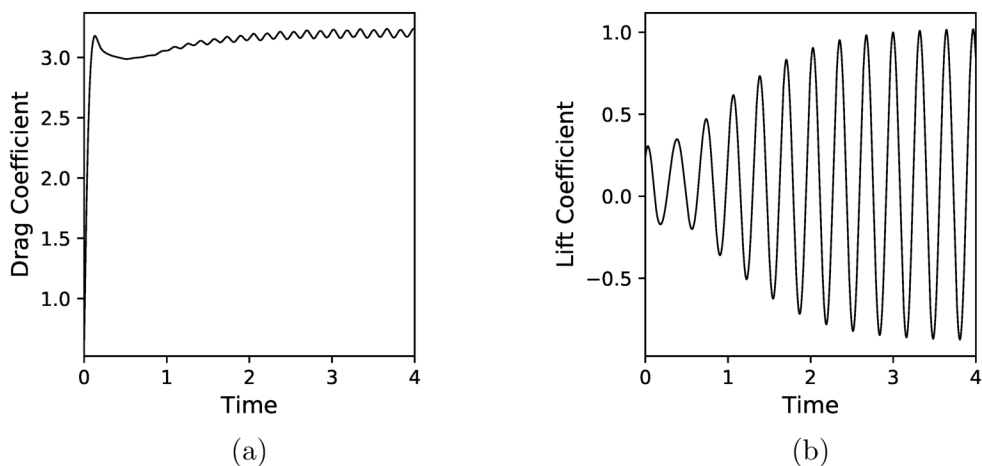
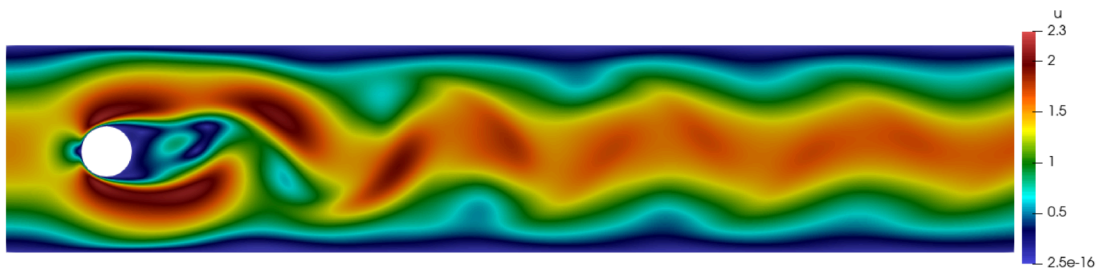


Figure 5.14: Plots of the (a) drag coefficient and (b) lift coefficient from initialization to steady vortex-shedding.

Figure 5.15a–b visualizes the velocity and pressure fields once steady vortex-shedding has been established.





(a)



(b)

Figure 5.15: Simulation results of (a) velocity and (b) pressure for the two dimensional Schäfer-Turek benchmark.

A similar three-dimensional benchmark is defined on the domain shown in figure 5.16. In this case, flow is around a rectangular prism instead of a cylinder.

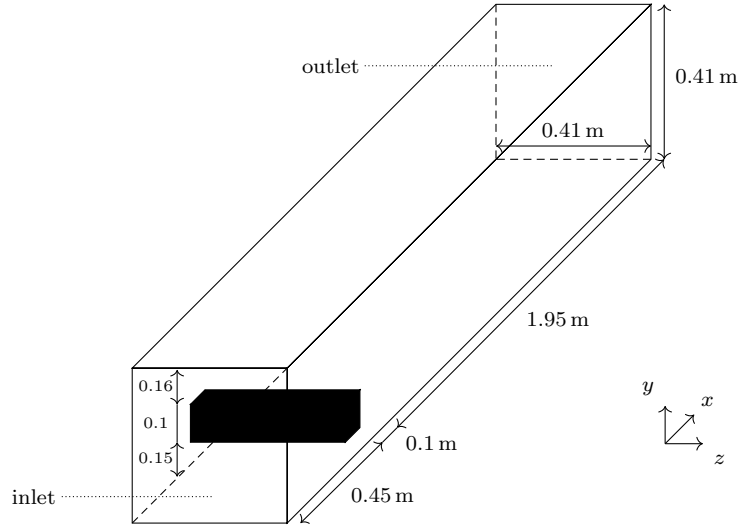


Figure 5.16: The simulation geometry for one of the three-dimensional Schäfer-Turek benchmarks.

The problem considered is unsteady flow with a time-varying inlet velocity (test case 3D-3Q [78]):

$$\mathbf{0} = \frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) - \nu \nabla^2 \mathbf{u} + \nabla p \quad \text{in } \Omega \quad (5.54)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \quad (5.55)$$

$$\mathbf{u} = \mathbf{0} \quad \text{on walls and object} \quad (5.56)$$

$$\mathbf{u} = \frac{36y(0.41 - y)z(0.41 - z)}{0.41^4} \boldsymbol{\delta}_x \quad \text{at inlet} \quad (5.57)$$

$$\mathbf{0} = \mathbf{n} \cdot (\mathbf{u}\mathbf{u} - \nu \nabla \mathbf{u} + p\mathbb{I}) - \max(\mathbf{u} \cdot \mathbf{n}, 0) \mathbf{u} \quad \text{at outlet} \quad (5.58)$$

A kinematic viscosity of  $1 \times 10^{-3}$  results in a time-varying Reynolds number in the range  $[0, 100]$ .

The same finite elements, polynomial order, time discretization scheme, and time step are used as for the two-dimensional benchmark but, in this case, a mesh with 1592 elements is used. The computed maximum values for the drag and lift coefficients, 4.6179 and 0.0428 respectively, again agree reasonably well with the literature values of 4.3000 – 4.5000 and 0.0100 – 0.0500 [78].

## 5.4 Multi-Component Flow

### 5.4.1 Multi-Component Diffusion

As a first example of multi-component flow, consider pure diffusion of some components A, B, and C:

$$\frac{\partial c_A}{\partial t} - \mathcal{D}_A \nabla^2 c_A = 0 \quad \text{in } \Omega \quad (5.59)$$

$$\frac{\partial c_B}{\partial t} - \mathcal{D}_B \nabla^2 c_B = 0 \quad \text{in } \Omega \quad (5.60)$$

$$\frac{\partial c_C}{\partial t} - \mathcal{D}_C \nabla^2 c_C = 0 \quad \text{in } \Omega \quad (5.61)$$

$$c_A(t = 0) = 1 \quad \text{in } \Omega \quad (5.62)$$

$$c_B(t = 0) = 0 \quad \text{in } \Omega \quad (5.63)$$

$$c_C(t = 0) = 0.5 \quad \text{in } \Omega \quad (5.64)$$

$$c_A = e^{-10t} \quad \text{on one wall} \quad (5.65)$$

$$c_B = 1 - e^{-10t} \quad \text{on one wall} \quad (5.66)$$

$$c_C = 0.5e^{-10t} \quad \text{on one wall} \quad (5.67)$$

$$-\mathbf{n} \cdot \nabla c_A = -\mathbf{n} \cdot \nabla c_B = -\mathbf{n} \cdot \nabla c_C = 0 \quad \text{on other walls} \quad (5.68)$$

The diffusion coefficients are  $\mathcal{D}_A = 1$ ,  $\mathcal{D}_B = 7 \times 10^{-1}$ , and  $\mathcal{D}_C = 3 \times 10^{-1}$ . After some time, the overall concentrations of the components in the domain are expected to plateau to zero for components A and C and one for component B following the concentration boundary conditions.

Error in the steady-state concentrations is given in table 5.3. An H1 finite element space with a polynomial order of three is used for each component concentration. Results are mesh- and time step-independent, so a mesh with 938 elements and a time step of  $\Delta t = 0.01$  s is used. All errors are very low as desired.

The time-evolution of the concentrations is shown in figure 5.17. As expected, given the different diffusion coefficients, the curve for component A plateaus the fastest followed by component B and then component C.

Table 5.3: Error results for multi-component diffusion.

Error Metric	Value
$c_A$ L2 Norm	$4.038 \times 10^{-27}$
$c_B$ L2 Norm	$4.544 \times 10^{-15}$
$c_C$ L2 Norm	$4.762 \times 10^{-9}$

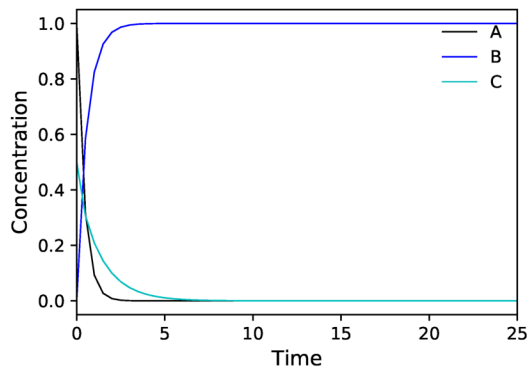


Figure 5.17: Simulation result of the concentrations of components A, B, and C for multi-component diffusion.

### 5.4.2 Multi-Component Convection and Diffusion

Convection and diffusion with a known velocity field is tested with a benchmark from Elman *et al* [36]:

$$\nabla \cdot (\mathbf{u}c_A) - \mathcal{D}_A \nabla^2 c_A = 0 \quad \text{in } x \in [-1, 1], y \in [-1, 1] \quad (5.69)$$

$$\mathbf{u} = \delta_y \quad \text{in } x \in [-1, 1], y \in [-1, 1] \quad (5.70)$$

$$c_A = -1 \quad \text{on } x = -1 \quad (5.71)$$

$$c_A = 1 \quad \text{on } x = 1 \quad (5.72)$$

$$c_A = x \quad \text{on } y = -1 \quad (5.73)$$

$$c_A = 0 \quad \text{on } y = 1 \quad (5.74)$$

which has the exact solution:

$$c_A = x \left( \frac{1 - e^{(y-1)/\mathcal{D}_A}}{1 - e^{-2/\mathcal{D}_A}} \right) \quad (5.75)$$

A diffusion coefficient of  $\mathcal{D}_A = 1 \times 10^{-2}$  is used, thus the problem is convection-dominated. An exponential boundary layer forms near  $y = 1$  with a width of  $e^{(1-y)/\mathcal{D}_A}$  [36].

Again an H1 finite element space is used for the concentration of component A, this time with a polynomial order of two. A mesh convergence test is conducted beginning with a very coarse mesh and, as shown in figure 5.18, the expected convergence rate is achieved. A higher convergence rate may be achieved with the discontinuous Galerkin method, but it is not yet implemented in `OpenCMP` for multi-component flows.

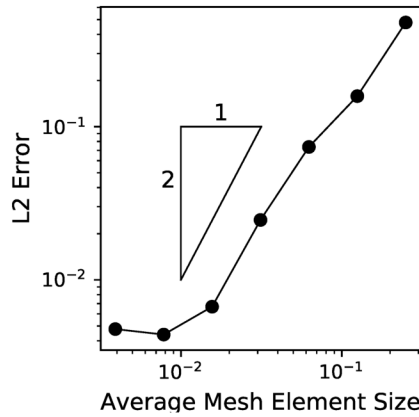


Figure 5.18: Mesh convergence for multi-component convection and diffusion.

The simulation output on the finest mesh is visualized in figure 5.19 and shows the sharp boundary layer transition.

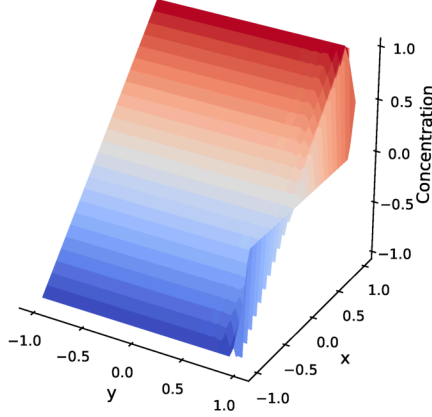


Figure 5.19: Simulation result of the concentration of component A for multi-component convection and diffusion.

### 5.4.3 Reacting Flow

To test reacting flow, consider channel flow convecting a component A, which also diffuses throughout the channel. Insert objects into the channel to mimic fixed catalyst particles and impose a first order reaction at the surfaces of these particles:



The transport and reaction of components A and B is described by the following equations:

$$\frac{\partial c_A}{\partial t} + \nabla \cdot (\mathbf{u}c_A) - \mathcal{D}_A \nabla^2 c_A = R_A \quad \text{in } \Omega \quad (5.77)$$

$$\frac{\partial c_B}{\partial t} + \nabla \cdot (\mathbf{u}c_B) - \mathcal{D}_B \nabla^2 c_B = -R_A \quad \text{in } \Omega \quad (5.78)$$

$$c_A(t=0) = 0 \quad \text{in } \Omega \quad (5.79)$$

$$c_B(t=0) = 0 \quad \text{in } \Omega \quad (5.80)$$

$$c_A = 1 \quad \text{at inlet} \quad (5.81)$$

$$c_B = 0 \quad \text{at inlet} \quad (5.82)$$

$$-\mathbf{n} \cdot \nabla c_A = -\mathbf{n} \cdot \nabla c_B = 0 \quad \text{on walls} \quad (5.83)$$

$$(\mathbf{n} \cdot \mathbf{u}) c_A - \mathcal{D}_A \mathbf{n} \cdot \nabla c_A - \max(\mathbf{n} \cdot \mathbf{u}, 0) c_A = 0 \quad \text{at outlet} \quad (5.84)$$

$$(\mathbf{n} \cdot \mathbf{u}) c_B - \mathcal{D}_B \mathbf{n} \cdot \nabla c_B - \max(\mathbf{n} \cdot \mathbf{u}, 0) c_B = 0 \quad \text{at outlet} \quad (5.85)$$

with:

$$R_A = -c_A \quad \text{at objects} \quad (5.86)$$

while the velocity and pressure fields are given by the solution of the incompressible Navier-Stokes equations:

$$\mathbf{0} = \frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) - \nabla^2 \mathbf{u} + \nabla p \quad \text{in } \Omega \quad (5.87)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \quad (5.88)$$

$$\mathbf{u}(t = 0) = \mathbf{0} \quad \text{in } \Omega \quad (5.89)$$

$$p(t = 0) = 0 \quad \text{in } \Omega \quad (5.90)$$

$$\mathbf{u} = 4y(0.5 - y)\boldsymbol{\delta}_x \quad \text{at inlet} \quad (5.91)$$

$$\mathbf{u} = \mathbf{0} \quad \text{on walls and objects} \quad (5.92)$$

$$\mathbf{0} = \mathbf{n} \cdot (\mathbf{u}\mathbf{u} - \nabla \mathbf{u} + p\mathbb{I}) - \max(\mathbf{u} \cdot \mathbf{n}, 0) \mathbf{u} \quad \text{at outlet} \quad (5.93)$$

The diffusion coefficients are  $\mathcal{D}_A = \mathcal{D}_B = 1$  and a kinematic viscosity of  $\nu = 1$  gives a Reynolds number of 0.125.

The Taylor-Hood finite element pair (VectorH1-H1) is used for the velocity and pressure with polynomial orders of two and one respectively. H1 elements of order two are used for components A and B. The nonlinear convection terms—for convection of the components and velocity—are linearized through the Oseen method to a relative error tolerance of  $1 \times 10^{-4}$  and absolute error tolerance of  $1 \times 10^{-6}$  or five iterations. The simulation is run for 10 s, to ensure a steady-state is reached, with a time step  $\Delta t = 1 \times 10^{-2}$  s. This problem has no exact solution so the results must be examined qualitatively. Figure 5.20 shows the concentration profiles of components A and B after 10 s. Component A enters at the inlet of the channel and is transported towards the objects. Its concentration quickly drops as it reacts at the surfaces of the objects. Component B shows the opposite, its concentration increasing along the length of the channel as it is produced at the surfaces of the objects.

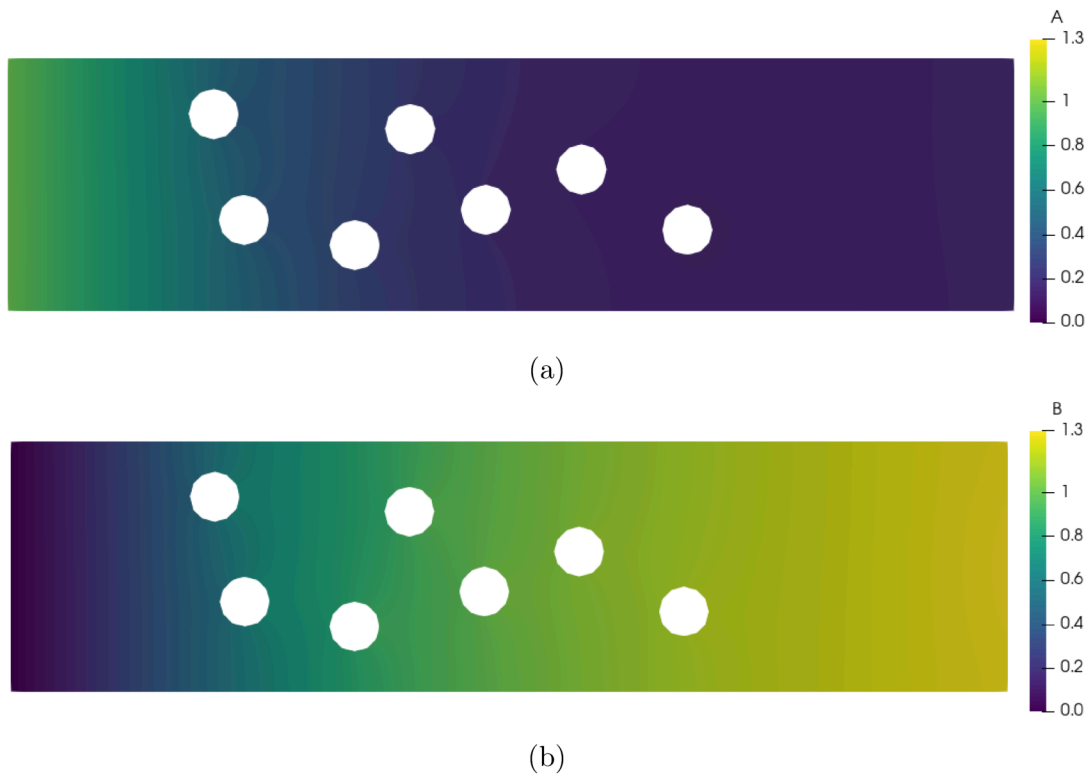


Figure 5.20: Simulation results of the concentrations of (a) component A and (b) component B for reacting flow.

Figure 5.21 shows the velocity and pressure fields after 10 s. The flow is parabolic at the inlet, due to the velocity Dirichlet boundary condition, and well-developed at the outlet. In the middle of the channel, the flow splits to form various paths around the fixed objects, with the highest velocities through the large path roughly along the middle of the channel. The relatively high pressure drop is indicative of the degree of obstruction to the flow.



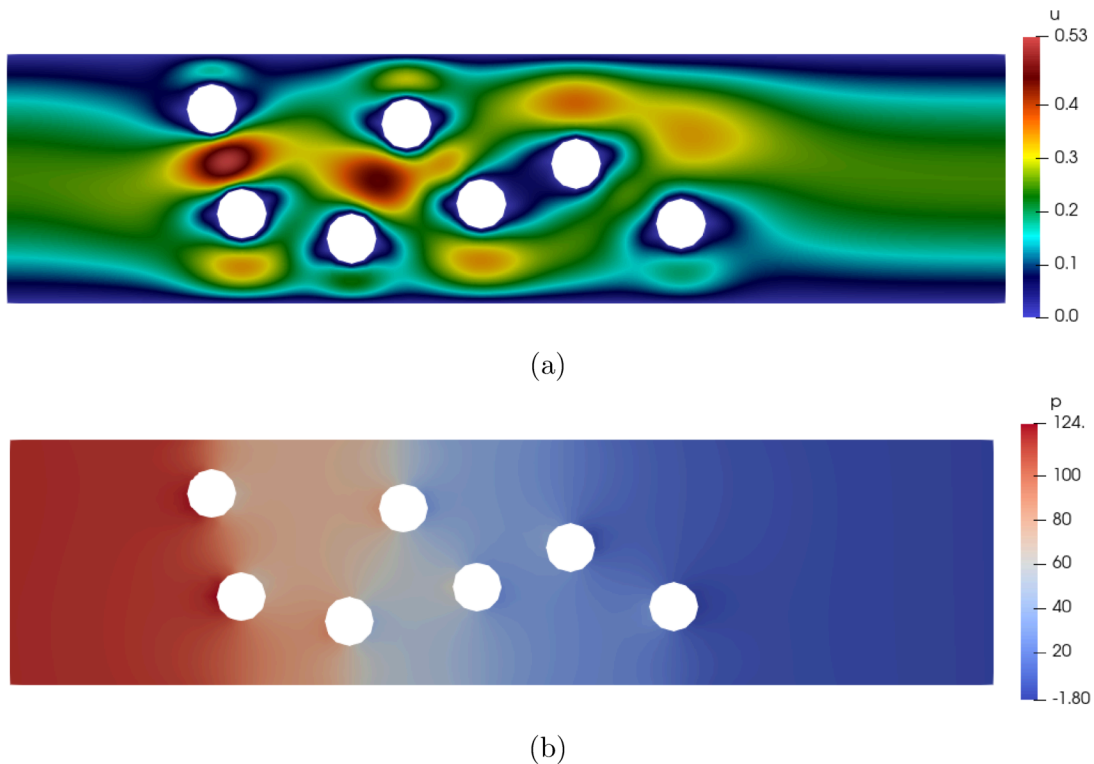


Figure 5.21: Simulation results of (a) velocity and (b) pressure for reacting flow.

## 5.5 Time Discretization Schemes

### 5.5.1 Fixed Time Step Schemes

Time step convergence is examined for all of the fixed time step time discretization schemes implemented (and fully working) in `OpenCMP` using the Taylor-Green vortices problem given in section 5.3.1. A refined mesh with 378 elements is used to ensure error is dominated by the temporal discretization error. In the case of implicit Euler and Crank-Nicolson, the nonlinear convection term is handled through Oseen-style linearization with a relative tolerance of  $1 \times 10^{-4}$  and an absolute tolerance of  $1 \times 10^{-6}$  or a maximum of five iterations. The three IMEX schemes all treat the convection term explicitly and the remaining terms implicitly. The discontinuous Galerkin method is used with the HDiv-L2 mixed finite element space in order to strongly enforce the incompressibility constraint. Because of this, the divergence of the velocity is close to machine precision for any time step and its

convergence is not considered. A polynomial order of three is used for the velocity and two for the pressure to satisfy the inf-sup condition.

Figure 5.22 shows the velocity and pressure results. All five time discretization schemes achieve their expected convergence rate for velocity. However, most schemes only achieve first-order convergence for pressure, even the second-order Crank-Nicolson and IMEX schemes. This is expected, as the time discretization of the incompressible Navier-Stokes equations effectively discretizes pressure with implicit Euler (first-order) regardless of the overall scheme chosen [79]. It is unclear why the third-order IMEX scheme shows third-order convergence for pressure, but it is likely a coincidence of the chosen problem. The third-order IMEX scheme did present unexpected challenges. The problem was defined on the  $[0, 2\pi] \times [0, 2\pi]$  domain used in section 5.3.1. However, the results for the third-order IMEX scheme are given for the domain  $[0, \pi] \times [0, \pi]$  as the scheme showed very poor convergence on the larger domain. The reason for this is unclear.

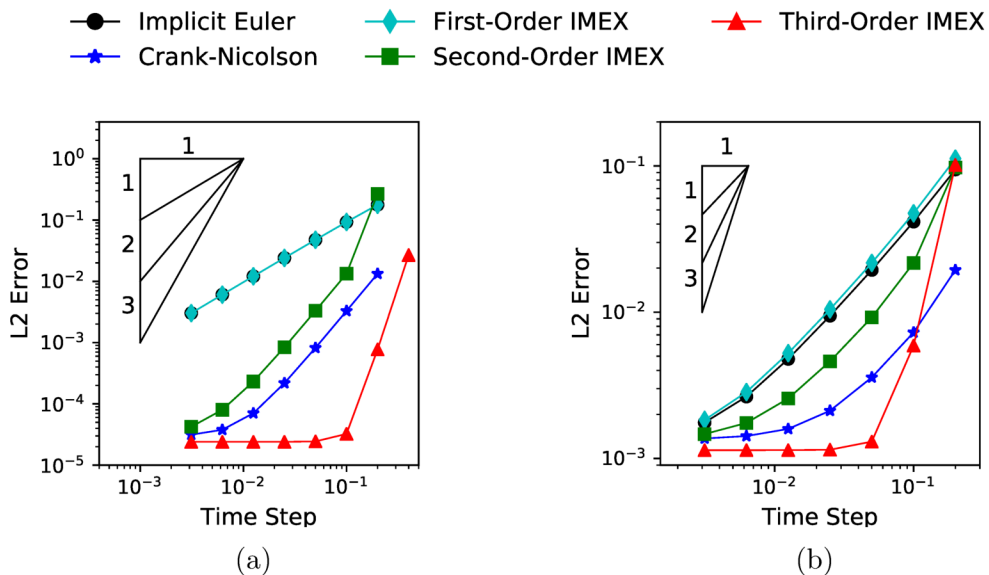


Figure 5.22: Time step convergence for (a) velocity and (b) pressure for the various available fixed time step time discretization schemes.

## 5.5.2 Adaptive Time-Stepping

Only one adaptive time-stepping scheme is implemented in `OpenCMP` and fully working so can be validated, the so-called adaptive two step scheme. At each time step, this scheme runs an implicit Euler solve (first-order) and a Crank-Nicolson solve (second-order). The local error is estimated based on the error between the first- and second-order results then is used to adjust the size of the time step to achieve a desired error tolerance. The final solution for the time step is taken from the implicit Euler solve so that the scheme remains unconditionally stable.

The adaptive time-stepping scheme is tested on the Poiseuille flow problem from section 5.2.1, but now with a time-varying inlet pressure boundary condition. This pressure boundary condition, and the velocity it produces, is shown in figure 5.23a. Its time evolution includes sharp jumps and long plateaus to force variation in the time step. A refined mesh with 128 elements is used to ensure fairly low spatial discretization error. The initial time step is  $\Delta t = 0.001$  s and the estimated local error is constrained to a relative tolerance of  $1 \times 10^{-4}$  and an absolute tolerance of  $1 \times 10^{-4}$ . The time step is also restricted to remain in the range  $[1 \times 10^{-10}, 1 \times 10^{-1}]$  to prevent either excessively small time steps or time steps too large to capture simulation dynamics.

The time evolution of the time step is shown in figure 5.23b and is seen to follow the evolution of the velocity and pressure fields. The time step is very small, on the order of  $1 \times 10^{-5}$ , during the step changes in the inlet pressure to ensure the local error constraint is met. However, once the inlet pressure plateaus, the time step increases rapidly up to the maximum allowed time step.

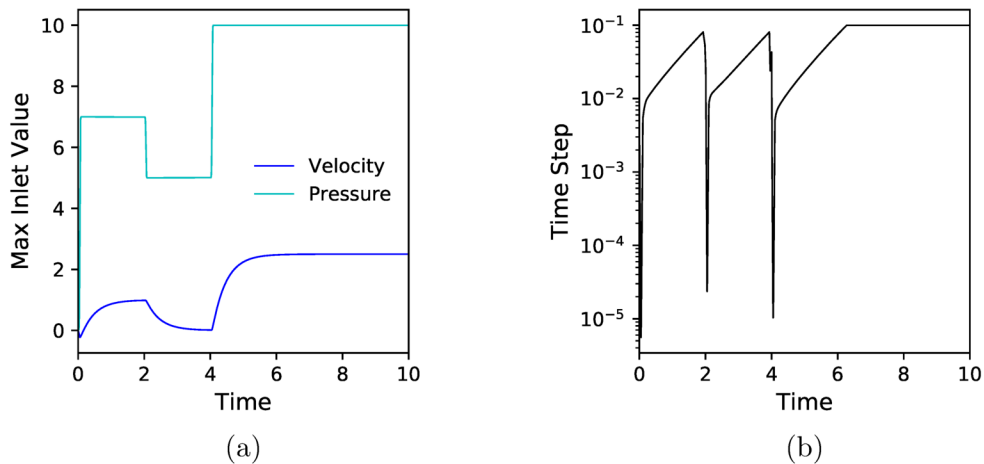


Figure 5.23: Plots of (a) the maximum inlet velocity and pressure and (b) the time step used by the adaptive time-stepping scheme.

## 5.6 Comparison to Existing Software

The previous sections have shown that `OpenCMP` performs to expected numerical accuracy and convergence rates. However, to be practically useful, `OpenCMP` must also show comparable simulation speed to other existing computational multiphysics software packages. This section compares `OpenCMP` to, likely the predominant open-source computational fluid dynamics software package, `OpenFOAM`<sup>®</sup> [20] on an example incompressible Navier-Stokes flow problem. More rigorous benchmarking on multiple models and against multiple existing software packages has yet to be carried out.

The test problem is the two-dimensional lid-driven cavity:

$$\nabla \cdot (\mathbf{u}\mathbf{u}) - \nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{0} \quad \text{in } x \in [0, 0.1], y \in [0, 0.1] \quad (5.94)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } x \in [0, 0.1], y \in [0, 0.1] \quad (5.95)$$

$$\mathbf{u} = \delta_x \quad \text{on } y = 0.1 \quad (5.96)$$

$$\mathbf{u} = \mathbf{0} \quad \text{on } y = 0 \quad (5.97)$$

$$\mathbf{u} = \mathbf{0} \quad \text{on } x = 0, 0.1 \quad (5.98)$$

using  $\nu = 2.5 \times 10^{-4}$  to give a Reynolds number of 400. This problem is given as one of the tutorials in the `OpenFOAM`<sup>®</sup> v8 user guide [73] and all configuration files used to run

the `OpenFOAM`<sup>®</sup> simulation in this comparison are taken from said user guide with some modifications to the exact physical properties and simulation parameters.

The `OpenFOAM`<sup>®</sup> simulation uses the `icoFoam` solver to run a transient solve to the steady-state solution of the lid-driven cavity. The initial conditions are as follows:

$$\mathbf{u}(t = 0) = \mathbf{0} \quad \text{in } x \in [0, 0.1], y \in [0, 0.1] \quad (5.99)$$

$$p(t = 0) = 0 \quad \text{in } x \in [0, 0.1], y \in [0, 0.1] \quad (5.100)$$

and the simulation runs for 2 s, as suggested by the user guide, with a time step of  $\Delta t = 1 \times 10^{-3}$  s.

The `OpenCMP` simulation directly solves the stationary problem through iterations of Oseen-style linearization. Convergence is specified as a relative tolerance of  $1 \times 10^{-6}$  and an absolute tolerance of  $1 \times 10^{-6}$  to match the residual tolerance of the `OpenFOAM`<sup>®</sup> simulation. `OpenCMP` simulations were carried out using both the standard finite element method with the Taylor-Hood mixed finite element space and using the discontinuous Galerkin method with the HDiv-L2 mixed finite element space. In both cases, the polynomial order was two for the velocity and one for the pressure to satisfy the inf-sup condition.

The original mesh was a uniform structured quadrilateral mesh with 1600 elements. Subsequent meshes were generated by refining this original mesh until approximately mesh-independent solutions were achieved for each numerical method. Time needed for mesh construction is not included in the overall timing comparison, but it was negligible in all cases.

The top half of table 5.4 compares the simulation time on the original mesh for the three simulations, all of which were run on the same personal computer under equivalent settings. `OpenFOAM`<sup>®</sup> is the fastest, though `OpenCMP` compares well when the standard finite element method is used. The discontinuous Galerkin method takes 30 – 50× longer than the other two simulations which is expected given the additional degrees of freedom and computational complexity of the numerical implementation. However, the discontinuous Galerkin method is also significantly more accurate on the original mesh than the other two simulations. Instead examining the mesh-independent results (the bottom half of table 5.4), the discontinuous Galerkin method is the fastest of the three numerical methods and remains slightly more accurate than either of the other two results. These results are expected and illustrate the benefits of including the discontinuous Galerkin method in `OpenCMP`. Compared to the finite volume method, the high-order polynomial interpolants of the discontinuous Galerkin method are able to accurately capture simulation dynamics without significant mesh refinement, lowering the computational costs for a given level of accuracy. The discontinuous Galerkin method also outperforms the standard finite element

method for this problem because the problem contains discontinuous boundary conditions that are challenging to model with a continuous trial function.

Table 5.4: Comparison to OpenFOAM®.

Package	Numerical Method	Solve Time	Mesh Elements	Error
OpenFOAM®	Finite Volume	3.94 s	1600	0.002889
OpenCMP	Finite Element	5.59 s	1600	0.004067
	Discontinuous Galerkin	210 s	1600	0.000592
OpenFOAM®	Finite Volume	256 s	25600	0.000876
OpenCMP	Finite Element	576 s	14400	0.000859
	Discontinuous Galerkin	210 s	1600	0.000592

Figure 5.24 compares the steady-state velocity streamlines of the three simulations on the original mesh. There is very little visible difference; all three show the expected center vortex, two lower corner vortices, and the beginnings of a vortex in the top left corner.

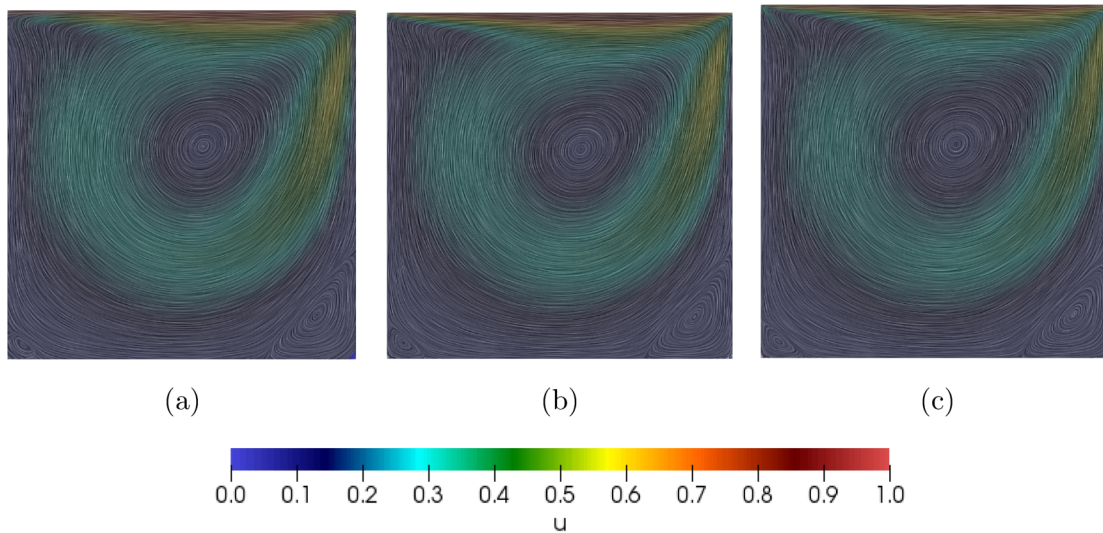


Figure 5.24: Simulation results for velocity from (a) OpenFOAM®, (b) OpenCMP using the standard finite element method, and (c) OpenCMP using the discontinuous Galerkin method for the lid-driven cavity.

Figure 5.25 compares the velocity results from the three simulations on the original

mesh to reference solutions given by Ghia *et al* [80]. All three results agree well with the reference solutions, but the improved accuracy of the discontinuous Galerkin method is evident, particularly at the local extrema.

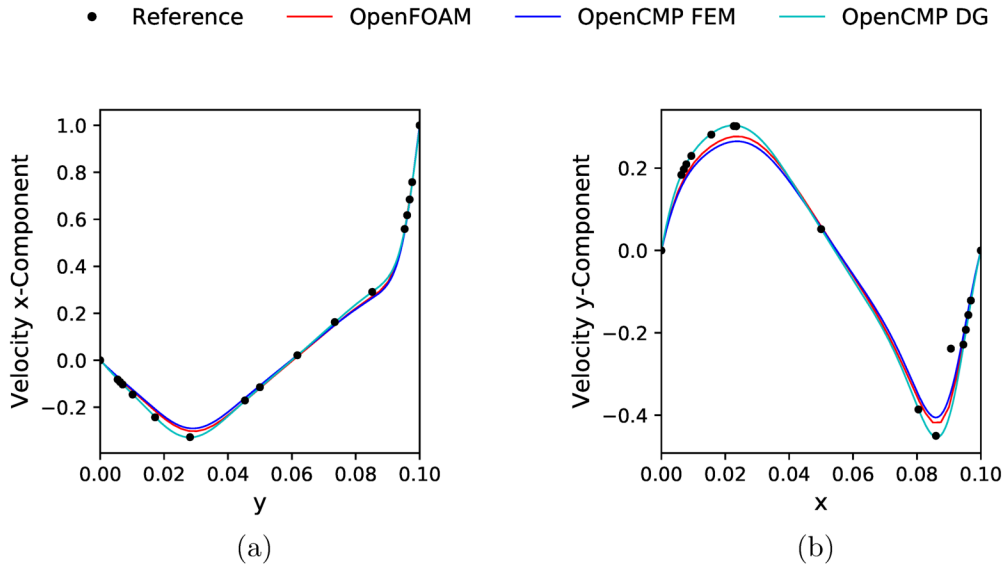


Figure 5.25: Comparison to the reference solutions from work by Ghia *et al* [80] of (a) the velocity x-component along  $y = 0.05$  and (b) the velocity y-component along  $x = 0.05$ .

# Chapter 6

## Extension of the Diffuse Interface Method

This chapter describes recent work on the diffuse interface method culminating in its use to model a rotating impeller. `OpenCMP` implements the diffuse interface method as described by Monte *et al* [8], including algorithms for automated phase field generation from CAD files and for applying multiple different boundary conditions along different regions of the same diffuse interface. A brief background on the diffuse interface method is also given in section 2.3. As discussed, the diffuse interface method offers the potential to speed up simulations by removing the need for time and labour intensive conformal meshing of complex geometries. It is particularly suited to preliminary design screening and similar applications where high accuracy is unnecessary. Monte *et al* [8] restricted their work to stationary domains. However, the diffuse interface method also has significant potential for applications with moving domains as the mesh no longer needs to move or deform to accommodate the domain movement. One such application, highly relevant to chemical engineering, is the simulation of impeller movement inside a chemical reactor.

### 6.1 Use of Impellers in Chemical Engineering

Figure 6.1 outlines a basic stirred tank reactor. The central shaft may hold several impellers and rotates to force mechanical mixing of the fluids inside the tank. Baffles on the walls of the tank break up the flow to prevent swirling and further assist axial mixing [81].

Mixing plays a key role in many chemical engineering applications. Improper mixing affects process yield and can cause problems during process scale-up. It can even affect the



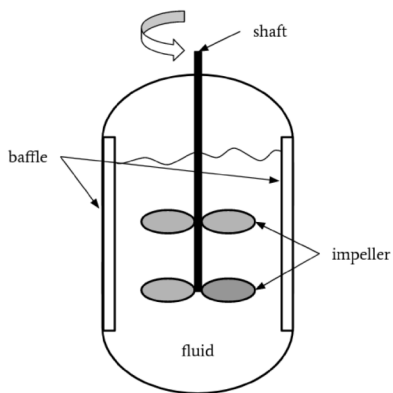


Figure 6.1: Diagram of a basic stirred tank batch reactor.

properties of the output product [81]. As such, it is important that simulations of chemical reactors accurately capture the mixing dynamics. In some cases, steady-state flow fields are known from experiment or can be approximated from a physical understanding of impeller effects. However, these are generally very simplistic models. For best accuracy it is necessary to include the actual impeller geometry and motion.

There are two main conformal mesh-based techniques for simulating impellers: the multiple reference frame approach and the sliding mesh approach. Both techniques conformally mesh the reactor tank with an inset impeller then divide said mesh into an inner region containing any rotating components, namely the shaft and impeller, and an outer region containing stationary components, namely the tank walls and baffles. The multiple reference frame technique models the fluid and solid components in the inner region in the rotating reference frame of the impeller by including the Coriolis force in the Navier-Stokes equations. The outer region is modelled in the standard stationary reference frame and a continuous velocity is enforced across the interface between the two mesh regions. The multiple reference frame approach is relatively computationally cheap and can be used for baffled tanks and multiple impellers moving at different speeds. However, it is only suitable for steady-state simulations and gives poor results when there is significant impeller-baffle interaction. The sliding mesh approach moves the inner mesh region relative to the outer mesh region to account for the impeller rotation. It is suitable for transient simulations and provides the highest mixing accuracy. However, the sliding mesh approach introduces significant computational complexity to move the mesh and to pass information between mesh regions in a conservative manner [81]. In summary, it is necessary to be able to accurately capture impeller movement for simulations of chemical reactors, but existing conformal mesh-based techniques suffer from low accuracy or high computational complexity due to

the intricacies of meshing a rotating impeller.

## 6.2 Phase Field Motion

Immersed boundary methods like the diffuse interface method have a long history with moving domains. Some of the first work on immersed boundary methods by Peskin [82] was used to model a human heart valve accounting for the movement of the valve leaflet to open and close the valve during a heartbeat. More recently, Aland *et al* [83] used a diffuse interface approach to model the movement and breakup of liquid droplets convected through a channel as well as the interface dynamics of a solid ball impacting a fluid. Sanders *et al* [84] used a level set method to model the settling and rising of discs of different buoyancies in a stationary fluid. Benk *et al* [85] validated an immersed boundary method on the FSI1 benchmark for fluid-structure interaction [86]. Most relevant to this work, Blais *et al* [87] used an immersed boundary method to simulate a pitched blade impeller in baffled and unbaffled tanks and obtained reasonable comparisons to experimental results and conformal mesh-based techniques.

Blais *et al* [87] added forcing terms to the incompressible Navier-Stokes equations within the fluid domain in order to mimic the force exerted by the impeller in the impeller-fluid interface region. A similar approach can be taken with the diffuse interface method. The fluid is modelled by the incompressible Navier-Stokes equations, restricted to the fluid domain by the use of the phase field (for the full equations see Appendix B). Then, the movement of the impeller is accounted for both by moving the phase field over time and by forcing the impeller-fluid interface to travel at the velocity of the impeller through Dirichlet boundary conditions at the diffuse interface (effectively a no-slip boundary condition).

This approach naturally requires a method for moving the phase field and determining its velocity. Sanders *et al* [84] suggest a method for updating the velocity of an immersed object at each time step which can account for the effects of fluid forces on the object. However, in the case of a stirred tank reactor, the impeller can be assumed to be a completely rigid body moving at a known, often constant, rotation speed. Thus, the phase field motion can be modelled by the equation of rigid body rotation [88]:

$$\mathbf{x} = \mathbf{R}(t)(\mathbf{X} - \mathbf{b}) + \mathbf{b} \tag{6.1}$$

where  $\mathbf{x}$  is the current location of the impeller given a starting location  $\mathbf{X}$ ,  $\mathbf{R}(t)$  is the rotation tensor, and rotation is about point  $\mathbf{b}$ . For a typical simulation of a stirred tank reactor,  $\mathbf{b}$  would simply be the origin.  $\mathbf{R}(t)$  would be as follows for a rotation speed of  $N$

revolutions per unit time and rotation about the z-axis:

$$\mathbf{R} = \begin{bmatrix} \cos(2\pi Nt) & -\sin(2\pi Nt) & 0 \\ \sin(2\pi Nt) & \cos(2\pi Nt) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.2)$$

The velocity at any point along the impeller would be as follows:

$$\mathbf{u}(x, y, z, t) = -2\pi Ny\boldsymbol{\delta}_x + 2\pi Nx\boldsymbol{\delta}_y \quad (6.3)$$

An example of phase field motion using eqn. (6.1) is shown in figure 6.2 for an ellipsoidal object with a rotation speed of  $N = 1$  RPS rotating about an offset point.

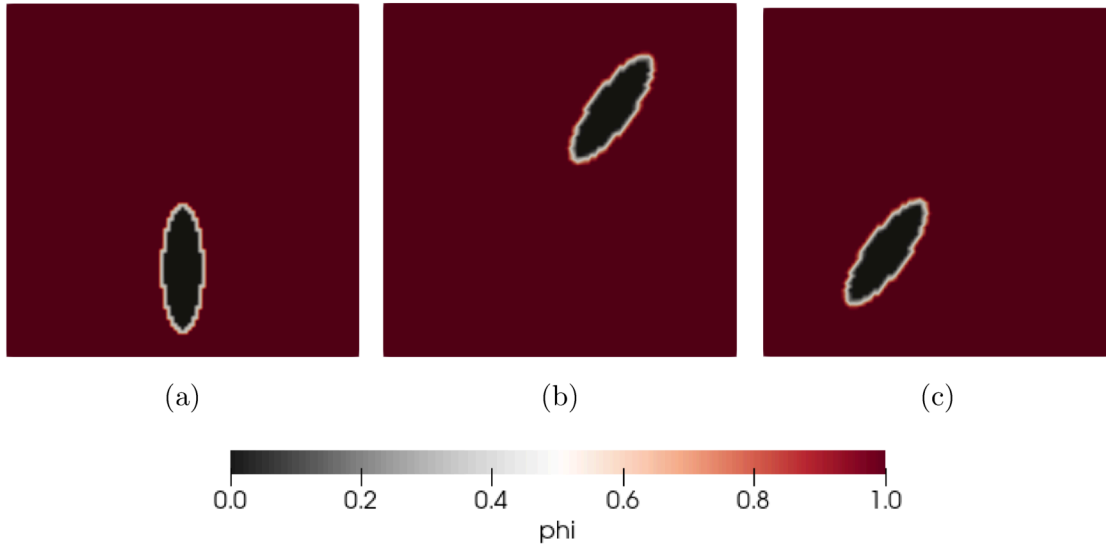


Figure 6.2: Phase field location at (a)  $t = 0$  s, (b)  $t = 0.4$  s, and (c)  $t = 0.9$  s. The phase field is one in the fluid domain and zero in the solid domain.

The final algorithm for the use of the diffuse interface method to simulate a moving rigid body within a flow field is as follows:

1. Determine the rotation tensor and axis of rotation for the rigid body motion.
2. Determine an analytical expression for the velocity of the rigid body.
3. Generate the phase field for the initial position of the rigid body.
4. At every time step:
  - 4.1. Update the location of the phase field using the initial phase field and eqn. (6.1).

- 4.2. Solve the diffuse interface formulation of the incompressible Navier-Stokes equations using the updated phase field and applying the velocity of the rigid body as a Dirichlet boundary condition along the diffuse interface.

For stability, it is also advisable to enforce that the fluid velocity inside the rigid body be equal to the velocity of the rigid body.

Figure 6.3 shows a two-dimensional test case using the same ellipsoidal object from figure 6.2 but now rotating about its center. The Taylor-Hood finite element pair (VectorH1-H1) is used with a polynomial order of two for the velocity and one for the pressure. The velocity and pressure fields are initialized as zero and the motion of the ellipse ramps from zero to a rotation speed of  $N = 0.16$  RPS over 0.25 s. In addition to the Dirichlet velocity boundary condition at the diffuse interface, a no-slip boundary condition is specified on the walls of the square domain. A time step of  $5 \times 10^{-4}$  s is used to ensure stability. The kinematic viscosity is taken to be  $\nu = 0.1$  and the ellipse length is 1.5 giving a Reynolds number of 3.6, highly laminar.

The velocity and pressure fields in figure 6.3 are given after the ellipse has completed two full rotations and, in both cases, the phase field was used to zero out the solid domain. The velocity field shows the expected rotational flow with vortices developing in the corners due to the use of a square domain. The pressure field also shows the expected regions of high pressure at the front edge of the ellipse and low pressure at the back edge (the ellipse rotates counter-clockwise).

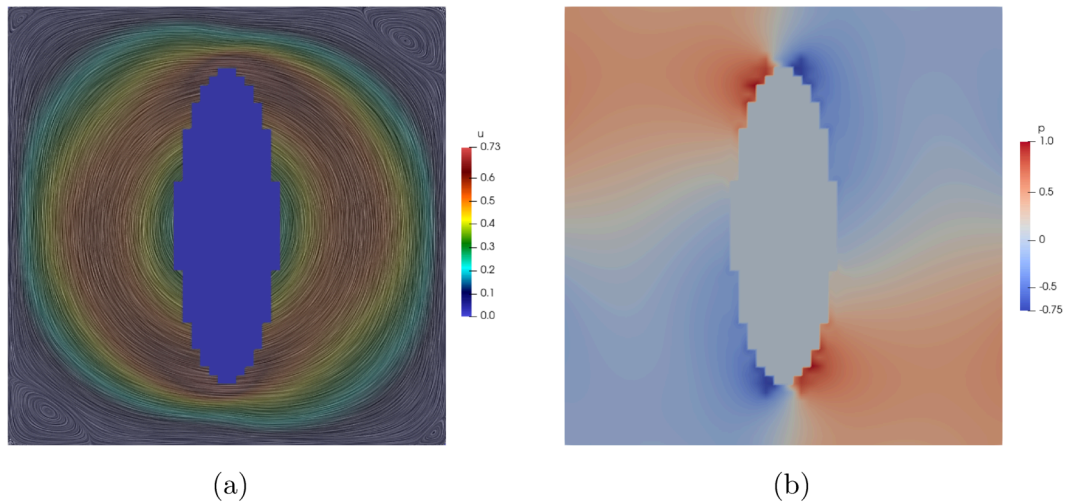


Figure 6.3: Simulation results of (a) velocity and (b) pressure for two-dimensional rigid body rotation with the diffuse interface method.

### 6.3 Diffuse Interface Simulation of a Stirred Tank Reactor

For verification of the application of the diffuse interface method to the simulation of impeller motion, this work considers the Rushton impeller shown in figure 6.4 and the tank geometry given in table 6.1 and shown in figure 6.4. The impeller design and geometry ratios were taken from experimental work by Bates *et al* [89] to enable comparison to said experimental results.

Table 6.1: Geometry of the stirred tank reactor and Rushton impeller.

Component	Geometry
Tank Height	12 cm
Tank Diameter	12 cm
Number of Baffles	4
Baffle Width	1 cm
Baffle Thickness	0.1 cm
Baffle Height	12 cm
Impeller Diameter	4 cm
Distance of Impeller from Tank Bottom	4 cm
Number of Impeller Blades	6
Impeller Blade Width	0.8 cm
Impeller Blade Thickness	0.1 cm
Impeller Blade Height	1 cm

Two dimensionless numbers can be used to predict the performance of a stirred tank reactor: the impeller Reynolds number and the Power number. These dimensionless numbers are defined as follows [81]:

$$Re = \frac{\rho N D^2}{\mu} \quad (6.4)$$

$$N_p = \frac{2\pi \mathbf{M}}{\rho N^2 D^5} \quad (6.5)$$

where  $N$  is the rotational speed of the impeller (RPS),  $D$  is the diameter of the impeller,  $\rho$  is the density of the fluid,  $\mu$  is the dynamic viscosity of the fluid, and  $\mathbf{M}$  is the torque acting on the impeller. The impeller Reynolds number predicts the flow regime inside

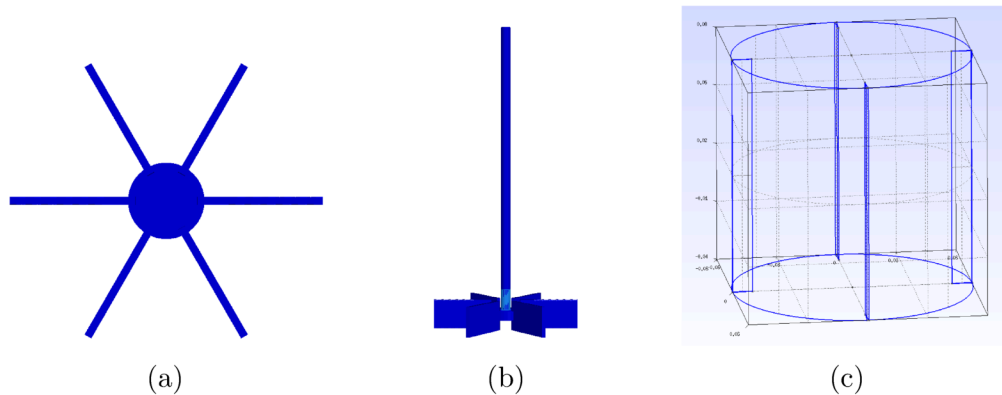


Figure 6.4: (a) top and (b) side view of the Rushton impeller and (c) tank geometry.

the reactor. Below  $Re = 10$  flow is laminar and fully turbulent flow is reached at  $Re = 1 \times 10^4$ . Impeller Reynolds numbers between these extremes indicate transitional flow [81]. The Power number relates the input impeller power (measured as torque) to the impeller Reynolds number. For Rushton impellers, Power number varies inversely proportional to impeller Reynolds number in the laminar flow regime but is constant regardless of impeller Reynolds number for fully turbulent flow [81].

In addition to the impeller geometry given in table 6.1, the diffuse interface test case considers a rotation speed of  $N = 1$  RPS and fluid properties of  $\rho = 1 \times 10^3$  kg/m<sup>3</sup> and  $\mu = 1 \times 10^{-3}$  Pa.s. This gives an impeller Reynolds number of 1600—within the transitional flow regime. For this impeller Reynolds number and the Rushton impeller used, experimental correlations provided by Bates *et al* [89] (primarily figure 1) predict a Power number of 3–4.

For the test case, the tank geometry shown in figure 6.4 is meshed by a tetrahedral mesh with 1605632 elements. A diffuse interface approximation to the impeller geometry is constructed on a smaller domain encompassing just the impeller then interpolated onto the main simulation mesh. No-slip boundary conditions were applied conformally on the tank walls and baffles while the impeller velocity boundary condition was applied at the diffuse interface as discussed in section 6.2. For speed, the Taylor-Hood finite element pair (VectorH1-H1) is used for velocity and pressure with polynomial interpolant orders of two and one respectively. The phase field uses the H1 finite element space with a polynomial interpolant order of two. A first-order IMEX scheme is used to linearize the incompressible Navier-Stokes equations and a time step of  $\Delta t = 1 \times 10^{-3}$  s is sufficient for stability. The velocity and pressure fields are initialized to zero and the impeller begins at a speed of 0 RPS then ramps up to the final speed of 1 RPS over 0.25 s. Unfortunately, the

interpolation scheme used to produce rigid body rotation of the phase field is very slow in three dimensions. Thus, it was not possible to run the simulation to a steady-state.

Figure 6.5 and figure 6.6 show velocity and pressure profiles after 0.86 s. Some degree of rotational flow is seen and regions of high and low pressure are seen at the front and back edges respectively of the impeller blades and baffles (the impeller rotates counter-clockwise). However, flow is clearly still developing; movement is only seen in the fluid close to the impeller blades and central shaft.

The Power number can be obtained from eqn. (6.5) with the torque on the impeller calculated as follows using the phase field to approximate the impeller geometry:

$$\mathbf{M} = \int_{\kappa} \mathbf{r} \times \left[ \nabla \phi \cdot \left( p\mathbb{I} - \mu (\nabla \mathbf{u} + (\nabla \mathbf{u})^T) \right) \right] dx \quad (6.6)$$

where  $\mathbf{r}$  is the position vector, in this case  $\mathbf{r} = x\boldsymbol{\delta}_x + y\boldsymbol{\delta}_y$ ,  $\mu$  is the dynamic viscosity, and the integral is a volume integral over the entire tank domain  $\kappa$ . The calculated Power number is 0.092 which is more than an order-of-magnitude lower than the expected value of 3–4 [89]. This is likely in part due to the undeveloped flow profile. It could also indicate inaccuracies in the phase field-based torque calculation. The phase field models the impeller surface as a diffuse region of finite thickness and includes this entire region in the torque calculation.



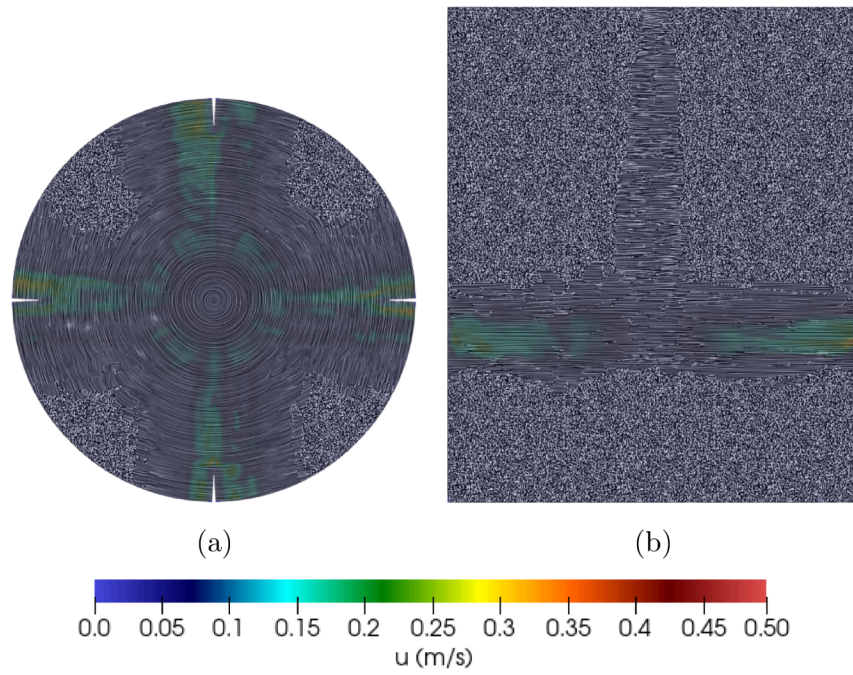


Figure 6.5: Simulation results for velocity in the (a) xy-plane and (b) xz-plane for a stirred tank reactor.



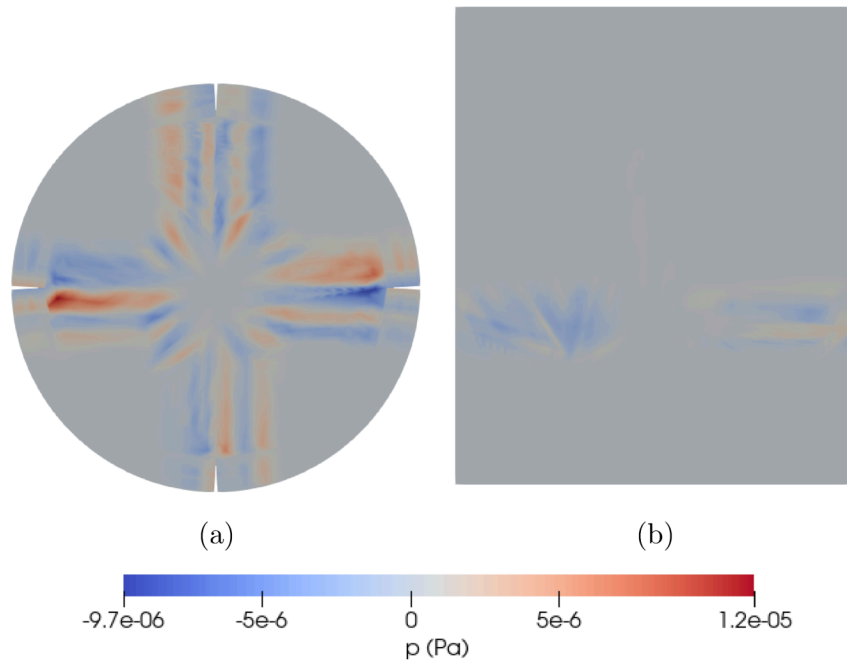


Figure 6.6: Simulation results for pressure in the (a) xy-plane and (b) xz-plane for a stirred tank reactor.

# Chapter 7

## Conclusions

### 7.1 Conclusions

This work presents `OpenCMP`, an open-source finite element-based computational multiphysics software package. The design of `OpenCMP` learns from and improves upon existing computational multiphysics software with a focus on high performance numerical methods, open access, ease of use, and a capacity for extensive customization. `OpenCMP` is validated on standard benchmarks and a short comparison—of speed and accuracy—is made to existing software. `OpenCMP` is also used to extend the diffuse interface method described by Monte *et al* [8] to problems with moving domains. Specifically, the diffuse interface method is used to simulate the rigid body rotation of an impeller in a stirred tank reactor.

The following lessons were learned from the `OpenCMP` design process:

- Consistent regular feedback from users during software development ensures the final software is easy to use and useful.
- For efficiency, the code back-end must use a compiled language. However, the addition of a `Python` [48] wrapper (or other high-level language) significantly increases code readability and aids development.
- It is more efficient to build off of existing finite element solver libraries than to implement the finite element method from scratch. The `NGSolve` finite element library [13] in particular offers high performance, access to meshing software and a graphical interface, and a `Python` wrapper and symbolic depictions of the finite element weak form for ease of development.
- A configuration file-based user interface provides the best trade-off between user friendliness and ease of development.

- Use of extensive back-end parsing tools allows a highly readable and intuitive configuration file format, including the ability to specify parameter values symbolically.
- A compartmentalized code structure allows models, time discretization schemes, and post-processing functionality to be modified or added without significant changes to the main code base.

The following conclusions can be drawn from the performance verification of `OpenCMP`:

- Expected mesh refinement error convergence rates are achieved both for the standard finite element method and the discontinuous Galerkin method for all models implemented in `OpenCMP`. Super-optimal convergence rates are achieved for some problems.
- Expected time step refinement error convergence rates are achieved for all time discretization schemes fully implemented in `OpenCMP`.
- Adaptive time-stepping constrains local error without requiring an excessively small time step for the full duration of the simulation.
- Using the standard finite element method, `OpenCMP` compares well to the popular computational fluid dynamics package `OpenFOAM`<sup>®</sup> [20] in speed and accuracy on a standard benchmark. The discontinuous Galerkin method is significantly slower but more accurate for a given mesh and polynomial interpolant order.

Finally, the following conclusions can be drawn regarding the application of the diffuse interface method to simulations with moving domains:

- The diffuse interface method can be used to simulate a moving domain through movement of the phase field without re-meshing or mesh movement.
- In the case of prescribed motion, such as rigid body rotation with a known rotation speed and axis of rotation, phase field motion can be solved independently then used to force movement of the fluid domain via a velocity Dirichlet boundary condition at the diffuse interface.

## 7.2 Recommendations

`OpenCMP` currently has a reasonably comprehensive feature set, particularly for laminar flow problems. However, the following additional functionality is required for it to compete with popular existing computational multiphysics software packages:

- Models for multi-phase flow and turbulent flow.
- Models for additional physics beyond fluid dynamics, in particular, heat transfer.
- Support for parallelization over multiple nodes with MPI [70].

- Built-in mesh generation capabilities.
- Built-in results visualization capabilities.

The following additions would also be desirable to improve the built-in capabilities of `OpenCMP`:

- More extensive post-processing including computation of user-specified design parameters.
- Adaptive mesh refinement based on local simulation error.

Finally, more rigorous comparison to other existing computational multiphysics software, particularly on computational efficiency, is desirable to justify the use of `OpenCMP` to the broader public.

The application of the diffuse interface method to the simulation of impeller motion in a stirred tank reactor could not be fully verified due to time constraints. The accuracy of fully developed flow must still be assessed. The diffuse interface method must also be evaluated against conformal mesh-based approaches—for accuracy and speed—to determine if it is a viable alternative. Finally, the current implementation of phase field motion is unreasonably slow and requires optimization.

# References

- [1] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis, “CFD vision 2030 study: A path to revolutionary computational aerosciences,” NASA, Langley Research Center, Hampton, VA, USA, techreport CR-2014-218178, Mar. 2014. [Online]. Available: <https://ntrs.nasa.gov/api/citations/20140003093/downloads/20140003093.pdf>
- [2] Z. L. Liu, *Multiphysics in Porous Materials*. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer International Publishing AG, 2018.
- [3] W. Yu, K. Zhang, and X. Li, “Recent algorithms on automatic hexahedral mesh generation,” in *The 10th International Conference on Computer Science & Education*, Fitzwilliam College, Cambridge University, UK, Jul. 2015.
- [4] J. Tu, G. H. Yeoh, and C. Liu, *Computational Fluid Dynamics: A Practical Approach*, 3rd ed. The Boulevard, Langford Lane, Kidlington, Oxford, UK: Butterworth-Heinemann, 2018.
- [5] C. G. Armstrong, H. J. Fogg, C. M. Tierney, and T. T. Robinson, “Common themes in multi-block structured quad/hex mesh generation,” *Procedia Eng.*, vol. 124, pp. 70–82, 2015.
- [6] R. Mittal and G. Iaccarino, “Immersed boundary methods,” *Annu. Rev. Fluid Mech.*, vol. 37, pp. 239–261, 2005.
- [7] S. K. F. Stoter, P. Müller, L. Cicalese, M. Tiveri, D. Schillinger, and T. J. R. Hughes, “A diffuse interface method for the Navier-Stokes/Darcy equations: Perfusion profile for a patient-specific human liver based on MRI scans,” *Comput. Methods Appl. Mech. Eng.*, vol. 321, pp. 70–102, 2017.

- [8] E. J. Monte, J. Lowman, and N. M. Abukhdeir, “A diffuse interface method for simulation-based screening of heat transfer processes with complex geometries,” 2021, arXiv:2101.06824.
- [9] N. Provatas and K. Elder, *Phase-Field Methods in Material Science and Engineering*. Wiley-VCH, 2010.
- [10] D. W. Pepper, S. Pirbastami, and D. B. Carrington, *50 Years of CFD in Engineering Sciences: A Commemorative Volume in Memory of D. Brian Spalding*. Singapore: Springer Singapore, 2020, ch. A Comparison Between FEM and FVM via the Method of Weighted Residuals, pp. 753–777.
- [11] B. Cockburn, G. E. Karniadakis, and C.-W. Shu, *Discontinuous Galerkin Methods: Theory, Computation and Applications*, 1st ed. Springer-Verlag, 2000, ch. The Development of Discontinuous Galerkin Methods, pp. 3–50.
- [12] “COMSOL Multiphysics<sup>®</sup>,” Online, COMSOL AB, Stockholm, Sweden. [Online]. Available: <https://www.comsol.com/>
- [13] J. Schöberl, “Netgen/NGSolve,” Online, 2020. [Online]. Available: <https://ngsolve.org/>
- [14] P. Bastian, F. Heimann, and S. Marnach, “Generic implementation of finite element methods in the distributed and unified numerics environment (DUNE),” *Kybernetika*, vol. 46, no. 2, pp. 294–315, 2010. [Online]. Available: <https://dml.cz/handle/10338.dmlcz/140745>
- [15] M. Malinen and P. Raback, *Multiscale Modelling Methods for Applications in Material Science*, ser. IAS Series. Forschungszentrum Juelich, Sep. 2013, vol. 19, ch. Elmer Finite Element Solver for Multiphysics and Multiscale Problems, pp. 101–113.
- [16] M. Alnaes, 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,” *J. Arch. Num. Soft.*, vol. 3, 2015.
- [17] R. Cimrman, V. Lukes, and E. Rohan, “Multiscale finite element calculations in Python using SfePy,” *Adv. Comput. Math.*, 2019.
- [18] P. Bastian, M. Blatt, A. Dedner, N. A. Dreier, C. Engwer, R. Fritze, C. Graser, C. Gruninger, D. Kempf, R. Kloforn, M. Ohlberger, and O. Sander, “The DUNE framework: Basic concepts and recent developments,” Jun. 2020, arXiv preprint.

Submitted to Computers & Mathematics with Applications. [Online]. Available: <https://arxiv.org/pdf/1909.13672.pdf>

- [19] A. Dedner, R. Kloforn, M. Nolte, and M. Ohlberger, “A generic interface for parallel and adaptive discretization schemes: abstraction principles and the DUNE-FEM module,” *Comput*, vol. 90, pp. 165–196, Aug. 2010.
- [20] “OpenFOAM <sup>®</sup>,” Online, OpenCFD Ltd. ESI Group. [Online]. Available: <https://www.openfoam.com/>
- [21] D. E. Keyes, L. C. McInnes, C. Woodward, W. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawlowski, A. P. Randles, D. Reynolds, B. Riviere, U. Rude, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, and B. Wohlmuth, “Multiphysics simulations: Challenges and opportunities,” *IJHPCA*, vol. 27, no. 1, 2013.
- [22] G. Jimeno, Y. C. Lee, and X. W. Ni, “Smoothed particle hydrodynamics: A new approach for modeling flow in oscillatory baffled reactors,” *Comput. Chem. Eng.*, vol. 124, pp. 14–27, May 2019.
- [23] A. Tamrakar and R. Ramachandran, “CFD-DEM-PBM coupled model development and validation of a 3D top-spray fluidized bed wet granulation process,” *Comput. Chem. Eng.*, vol. 125, pp. 249–270, Jun. 2019.
- [24] C. Casado, J. Marugán, R. Timmers, M. M. noz, and R. van Grieken, “Comprehensive multiphysics modeling of photocatalytic process by computational fluid dynamics based on intrinsic kinetic parameters determined in a differential photoreactor,” *Chem. Eng. J.*, vol. 310, no. 2, pp. 368–380, Feb. 2017.
- [25] A. R. Rajabzadeh, R. L. Legge, and K. P. Weber, “Multiphysics modelling of flow dynamics, biofilm development and wastewater treatment in a subsurface vertical flow constructed wetland mesocosm,” *Ecol. Eng.*, vol. 74, pp. 107–116, Jan. 2015.
- [26] M. Ghadiri, A. Hemmati, A. T. Nakhjiri, and S. Shirazian, “Modelling tyramine extraction from wastewater using a non-dispersive solvent extraction process,” *Environ. Sci. Pollut. Res.*, vol. 27, pp. 39 068–39 076, Jul. 2020.

- [27] R. B. Bird, W. E. Stewart, and E. N. Lightfoot, *Transport Phenomena*, 2nd ed. 605 Third Avenue, New York, NY, USA: John Wiley & Sons Inc., 2002.
- [28] J. H. Ferziger and M. Perić, *Computational Methods for Fluid Dynamics*, 3rd ed. Springer-Verlag, 2002.
- [29] R. G. Rice and D. D. Do, *Applied Mathematics and Modeling for Chemical Engineers*, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2012.
- [30] J. D. Hoffman, *Numerical Methods for Engineers and Scientists*, 2nd ed. Hutgasse 4, Postfach 812, CH-4001 Basel, Switzerland: Marcel Dekker, 2001, ch. 4, pp. 187–243.
- [31] S. Larsson and V. Thomée, *Partial Differential Equations with Numerical Methods*, ser. Texts in Applied Mathematics. Berlin, Germany: Springer-Verlag, 2009, vol. 45, ch. 12, pp. 201–216.
- [32] B. Cockburn, G. Kanschat, D. Schötzau, and C. Schwab, “Local discontinuous Galerkin methods for the Stokes system,” *SIAM J. Numer. Anal.*, vol. 40, no. 1, pp. 319–343, Apr. 2002.
- [33] B. Cockburn, G. Kanschat, and D. Schötzau, “The local discontinuous Galerkin method for the Oseen equations,” *Math. Comput.*, vol. 73, no. 246, pp. 569–593, May 2003.
- [34] —, “A locally conservative LDG method for the incompressible Navier-Stokes equations,” *Math. Comput.*, vol. 74, no. 251, pp. 1067–1095, Oct. 2004.
- [35] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini, “Unified analysis of discontinuous Galerkin methods for elliptic problems,” *SIAM J. Numer. Anal.*, vol. 39, no. 5, pp. 1749–1779, 2002.
- [36] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics*, ser. Numerical Mathematics and Scientific Computation. Great Clarendon Street, Oxford, UK: Oxford University Press, 2014.
- [37] C. Li, S. Qiang, and X. Hua, “Advances in automatic hexahedral meshing,” *J. Phys. Conf. Ser.*, vol. 1637, 2020.
- [38] L. H. Nguyen, S. K. F. Stoter, M. Ruess, M. A. S. Uribe, and D. Schillinger, “The diffuse Nitsche method: Dirichlet constraints on phase-field boundaries,” *Int. J. Numer. Methods Eng.*, vol. 113, pp. 601–633, 2018.



- [39] L. Nguyen, S. Stoter, T. Baum, J. Kirschke, M. Ruess, Z. Yosibash, and D. Schillinger, “Phase-field boundary conditions for the voxel finite cell method: Surface-free stress analysis of CT-based bone structures,” *Int. J. Numer. Methods Biomed. Eng.*, vol. 33, p. e2880, 2017.
- [40] D. Rempfer, “On boundary conditions for incompressible navier-stokes problems,” *Appl. Mech. Rev.*, vol. 59, pp. 107–125, May 2006.
- [41] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso, “SU2: An open-source suite for multiphysics simulation and design,” *AIAA J.*, vol. 54, no. 3, Mar. 2016.
- [42] T. D. Economon, “SU2 configuration file: Transonic inviscid flow around a NACA0012 airfoil,” Online, Jun. 2014. [Online]. Available: [https://github.com/su2code/SU2/blob/master/QuickStart/inv\\_NACA0012.cfg](https://github.com/su2code/SU2/blob/master/QuickStart/inv_NACA0012.cfg)
- [43] —, “SU2 configuration file,” Online, 2014. [Online]. Available: [https://github.com/su2code/SU2/blob/master/config\\_template.cfg](https://github.com/su2code/SU2/blob/master/config_template.cfg)
- [44] ISO, *ISO/IEC 14882:2020(E) – Programming Language C++*. pub-ISO, 2020. [Online]. Available: <https://isocpp.org/std/the-standard>
- [45] D. Arndt, W. Bangerth, B. Blais, T. C. Clevenger, M. Fehling, A. V. Grayver, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J. P. Pelteret, R. Rastak, I. Thomas, B. Turcksin, Z. Wang, and D. Wells, “The deal.II library, version 9.2,” *J. Numer. Math.*, vol. 28, no. 3, 2020.
- [46] C. J. Permann, D. R. Gaston, D. Andrš, R. W. Carlsen, F. Kong, A. D. Lindsay, J. M. Miller, J. W. Peterson, A. E. Slaughter, R. H. Stogner, and R. C. Martineau, “MOOSE: Enabling massively parallel multiphysics simulation,” *SoftwareX*, vol. 11, p. 100430, 2020.
- [47] F. Hecht, “New development in FreeFem++,” *J. Numer. Math.*, vol. 20, no. 3–4, pp. 251–265, 2012.
- [48] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA, USA: CreateSpace, 2009.
- [49] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *SIAM Rev.*, vol. 59, no. 1, 2017.

- [50] “Ansys<sup>®</sup> Fluent,” Online, ANSYS Inc. [Online]. Available: <https://www.ansys.com/products/fluids/ansys-fluent>
- [51] “Ansys<sup>®</sup> CFX,” Online, ANSYS Inc. [Online]. Available: <https://www.ansys.com/products/fluids/ansys-cfx>
- [52] “Simcenter STAR-CCM+,” Online, Siemens. [Online]. Available: <https://www.plm.automation.siemens.com/global/en/products/simcenter/STAR-CCM.html>
- [53] “SimScale,” Online, SimScale. [Online]. Available: <https://www.simscale.com/>
- [54] “FEATool Multiphysics<sup>™</sup>,” Online, Precise Simulation. [Online]. Available: <https://www.featool.com/>
- [55] “Code\_Aster: Structures and thermomechanics analysis for studies and research,” Online, Electricité de France. [Online]. Available: <https://www.code-aster.org>
- [56] G. Dhondt and K. Wittig, “CalculiX: A free software three-dimensional structural finite element program,” Online, CalculiX. [Online]. Available: <http://www.calculix.de/>
- [57] “MATLAB,” Online, The Mathworks Inc., Natick, Massachusetts. [Online]. Available: <https://www.mathworks.com/products/matlab.html>
- [58] “OpenFOAM,” Online, CFD Direct Ltd. [Online]. Available: <https://cfd.direct/>
- [59] “FOAM-Extend,” Online, Wikki Ltd. [Online]. Available: <http://wikki.gridcore.se/foam-extend>
- [60] “GUI,” Online, Wikki Ltd. [Online]. Available: <https://openfoamwiki.net/index.php/GUI>
- [61] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, “libMesh: A C++ library for parallel adaptive mesh refinement/coarsening simulations,” *Eng. Comput.*, vol. 22, no. 3-4, pp. 237–254, 2006.
- [62] M. S. Alnaes, A. Logg, K. B. Olgaard, M. E. Rognes, and G. N. Wells, “Unified form language: A domain-specific language for weak formulations of partial differential equations,” *ACM Trans. Math. Software*, vol. 40, no. 9, Mar. 2014.
- [63] S. Badia and F. Verdugo, “Gridap: An extensible finite element toolbox in Julia,” *J. Open Source Softw.*, vol. 5, no. 52, p. 2520, 2020.

- [64] C. Geuzaine and J.-F. Remacle, “Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities,” *Int. J. Numer. Methods Eng.*, vol. 79, no. 11, pp. 1309–1331, 2009.
- [65] A. Ribes and C. Caremoli, “Salome platform component model for numerical simulation,” in *COMPSAC 07: Proceeding of the 31st Annual International Computer Software and Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 553–564.
- [66] J. Ahrens, B. Geveci, and C. Law, “ParaView: An end-user tool for large data visualization,” Los Alamos National Laboratory, Los Alamos, NM, USA, techreport LA-UR-03-1560, 2005.
- [67] F. S. F. Inc., “GNU lesser general public license v2.1,” Online, Feb. 1999. [Online]. Available: <https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>
- [68] 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, Í. 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,” *Nat. Methods*, vol. 17, pp. 261–272, 2020.
- [69] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, “PETSc Web page,” Online, 2021. [Online]. Available: <https://www.mcs.anl.gov/petsc>
- [70] M. P. I. Forum, *MPI: A Message-Passing Interface Standard Version 3.1*. University of Stuttgart, Stuttgart, Germany: High-Performance Computing Center, 2015.
- [71] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. 3600 Market Street, Floor 6, Philadelphia, PA, USA: SIAM, 1998.
- [72] “configparser: Configuration file parser,” Online. [Online]. Available: <https://docs.python.org/3/library/configparser.html>

- [73] T. O. Foundation, *OpenFOAM v8 User Guide*. [Online]. Available: <https://cfd.direct/openfoam/user-guide>
- [74] P. T. McGuire, “PyParsing,” Online, 2020. [Online]. Available: <https://pyparsing-docs.readthedocs.io/en/latest/> V3.0.0.
- [75] T. D. Economon and T. Albring, “The SU2 tutorial collection,” Online, Nov. 2019, for version 7.
- [76] D. N. Arnold and A. Logg, “Periodic table of the finite elements,” *SIAM News*, vol. 47, no. 9, Nov. 2014. [Online]. Available: <https://www-users.cse.umn.edu/~arnold/femtable/>
- [77] G. I. T. F. R. S., “LXXV. on the decay of vortices in a viscous fluid,” *Philos. Mag.*, vol. 46, no. 274, pp. 671–674, 1923, series 6.
- [78] M. Schäfer and S. Turek, “Benchmark computations of laminar flow around a cylinder,” in *Flow Simulation with High-Performance Computers II*, ser. Notes on Numerical Fluid Mechanics, E. H. Hirschel, Ed. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, 1996, vol. 52, ch. 4, pp. 547–566.
- [79] V. John, G. Matthies, and J. Rang, “A comparison of time-discretization/linearization approaches for the incompressible Navier-Stokes equations,” *Comput. Methods Appl. Mech. Eng.*, vol. 195, no. 44–47, pp. 5995–6010, Sep. 2006.
- [80] U. Ghia, K. N. Ghia, and C. T. Shin, “High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method,” *J. Comput. Phys.*, vol. 48, no. 3, pp. 387–411, 1982.
- [81] E. L. Paul, V. A. Atiemo-Obeng, and S. M. Kresta, Eds., *Handbook of Industrial Mixing: Science and Practice*. Hoboken, NJ, USA: John Wiley & Sons Inc., 2004.
- [82] C. S. Peskin, “Flow patterns around heart valves: A numerical method,” *J. Comput. Phys.*, vol. 10, pp. 252–271, 1972.
- [83] S. Aland, J. Lowengrub, and A. Voigt, “Two-phase flow in complex geometries: A diffuse domain approach,” *Comput. Model Eng. Sci.*, vol. 57, pp. 77–106, 2010.
- [84] J. Sanders, J. E. Dolbow, P. J. Mucha, and T. A. Laursen, “A new method for simulating rigid body motion in incompressible two-phase flow,” *Int. J. Numer. Methods Fluids*, vol. 67, pp. 713–732, 2011.

- [85] J. Benk, M. Ulbrich, and M. Mehl, “The Nitsche method of the Navier-Stokes equations for immersed and moving boundaries,” in *Proceedings of the Seventh International Conference on Computational Fluid Dynamics, ICCFD7. International Conference on Computational Fluid Dynamics*. Big Island, Hawaii: None, Jul. 2012.
- [86] J. Hron and S. Turek, *Fluid-Structure Interaction*, ser. Lecture Notes in Computational Science and Engineering. Springer-Verlag, 2006, vol. 53, ch. Proposal for Numerical Benchmarking of Fluid-Structure Interaction between Elastic Objects and Laminar Incompressible Flow, pp. 371–385.
- [87] B. Blais, M. Lassaigne, C. Goniva, L. Fradette, and F. Bertrand, “A semi-implicit immersed boundary method and its application to viscous mixing,” *Comput. Chem. Eng.*, vol. 85, pp. 136–146, 2016.
- [88] M. Lai, E. Krempl, and D. Ruben, *Introduction to Continuum Mechanics*, 4th ed. 30 Corporate Drive, Suite 400, Burlington, MA, USA: Elsevier, 2010, ch. 3, p. 82.
- [89] R. L. Bates, P. L. Fondy, and R. R. Corpstein, “An examination of some geometric parameters of impeller power,” *Ind. Eng. Chem.*, vol. 2, no. 4, pp. 310–314, 1963.
- [90] S. Rhebergen and G. N. Wells, “A hybridizable discontinuous Galerkin method for the Navier-Stokes equations with pointwise divergence-free velocity field,” *J. Sci. Comput.*, vol. 76, pp. 1484–1501, Feb. 2018.
- [91] U. M. Ascher, S. J. Ruuth, and B. T. R. Wetton, “Implicit-explicit methods for time-dependent partial differential equations,” *SIAM J. Numer. Anal.*, vol. 32, no. 3, pp. 797–823, Jun. 1995.
- [92] V. DeCaria and M. Schneier, “An embedded variable step IMEX scheme for the Navier-Stokes equations,” *Comput. Methods Appl. Mech. Eng.*, vol. 376, 2021.
- [93] J. Nitsche, “Über ein variationsprinzip zur lösung von Dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind,” *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, vol. 36, no. 1, pp. 9–15, Jul 1971. [Online]. Available: <https://doi.org/10.1007/BF02995904>

# APPENDICES

# Appendix A

## Discontinuous Galerkin Formulations of OpenCMP Models

The following derivations were produced with the help of James Lowman and Alex Vasile based on a similar derivation by Arnold *et al* [35].

In all of the derivations the gradient and double inner product operators are defined as follows:

$$\nabla \phi = \sum_i \sum_j \frac{\partial \phi_i}{\partial x_j} \delta_i \delta_j \quad (\text{A.1})$$

$$\gamma : \eta = \sum_i \sum_j \gamma_{ij} \eta_{ij} \quad (\text{A.2})$$

to agree with the conventions of the `NGSolve` finite element library [13].

### A.1 The Poisson Equation

Consider a domain  $\Omega$  with boundary  $\Gamma$  meshed by a triangulation  $\mathcal{T}$  of mesh elements  $\mathcal{K}$ . The Poisson equation is defined as follows with possible Dirichlet, Neumann, and Robin

boundary conditions on mesh boundary regions  $\Gamma_D$ ,  $\Gamma_N$ , and  $\Gamma_R$  respectively:

$$-\nabla^2 u = f \quad \text{in } \Omega \quad (\text{A.3})$$

$$u = g \quad \text{on } \Gamma_D \quad (\text{A.4})$$

$$-\mathbf{n} \cdot \nabla u = h \quad \text{on } \Gamma_N \quad (\text{A.5})$$

$$\mathbf{n} \cdot \nabla u = r(u - q) \quad \text{on } \Gamma_R \quad (\text{A.6})$$

where  $f$  is some source function.

Define  $\boldsymbol{\sigma}$  to be the gradient of  $u$  and use it to reduce this second order partial differential equation into a redundant system of first order partial differential equations:

$$\boldsymbol{\sigma} = \nabla u \quad (\text{A.7})$$

$$-\nabla \cdot \boldsymbol{\sigma} = f \quad (\text{A.8})$$

Insert trial functions  $\boldsymbol{\sigma}_A$  and  $u_A$  to form residuals and average with weighting functions  $\boldsymbol{\tau}$  for eqn. (A.7) and  $v$  for eqn. (A.8):

$$\int_{\Omega} \boldsymbol{\tau} \cdot \boldsymbol{\sigma}_A \, dx - \int_{\Omega} \boldsymbol{\tau} \cdot \nabla u_A \, dx = R_{\sigma} \quad (\text{A.9})$$

$$- \int_{\Omega} v (\nabla \cdot \boldsymbol{\sigma}_A) \, dx - \int_{\Omega} v f \, dx = R_u \quad (\text{A.10})$$

Apply the product rule and then the divergence theorem to eqn. (A.9):

$$\int_{\Omega} \boldsymbol{\tau} \cdot \boldsymbol{\sigma}_A \, dx - \int_{\Omega} \nabla \cdot (u_A \boldsymbol{\tau}) \, dx + \int_{\Omega} u_A (\nabla \cdot \boldsymbol{\tau}) \, dx = R_{\sigma} \quad (\text{A.11})$$

$$\int_{\Omega} \boldsymbol{\tau} \cdot \boldsymbol{\sigma}_A \, dx - \int_{\Gamma} \mathbf{n} \cdot (u_A \boldsymbol{\tau}) \, ds + \int_{\Omega} u_A (\nabla \cdot \boldsymbol{\tau}) \, dx = R_{\sigma} \quad (\text{A.12})$$

and to eqn. (A.10):

$$\int_{\Omega} \boldsymbol{\sigma}_A \cdot \nabla v \, dx - \int_{\Omega} \nabla \cdot (v \boldsymbol{\sigma}_A) \, dx - \int_{\Omega} v f \, dx = R_u \quad (\text{A.13})$$

$$\int_{\Omega} \boldsymbol{\sigma}_A \cdot \nabla v \, dx - \int_{\Gamma} \mathbf{n} \cdot (v \boldsymbol{\sigma}_A) \, ds - \int_{\Omega} v f \, dx = R_u \quad (\text{A.14})$$

Substituting eqn. (A.12) into eqn. (A.14) produces the standard finite element method weak form. Using the discontinuous Galerkin method instead, the trial functions  $u_A$  and  $\boldsymbol{\sigma}_A$  are allowed to be discontinuous across mesh element boundaries. Integrals over the



entire simulation domain  $\Omega$  become the sum of integrals over individual mesh elements  $\mathcal{K}$ . Furthermore, integrals over the simulation domain boundary  $\Gamma$  now include all mesh element boundaries  $\partial\mathcal{K}$ . The degeneracy of the trial functions on mesh element boundaries poses a problem for boundary integral terms as they may evaluate to different values within each of the two mesh elements that share any given mesh element boundary. To resolve this, new variables are defined which are single-valued on mesh element boundaries. These variables,  $\hat{u}$  and  $\hat{\sigma}$ , will be referred to as facet values. They are functions of  $u_A$  and  $\sigma_A$  and their specific definitions will be given later in this section. Accounting for this, eqn. (A.12) and eqn. (A.14) become as follows:

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\tau} \cdot \boldsymbol{\sigma}_A dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \mathbf{n} \cdot (\hat{u}\boldsymbol{\tau}) ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} u_A (\boldsymbol{\nabla} \cdot \boldsymbol{\tau}) dx = R_\sigma \quad (\text{A.15})$$

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A \cdot \boldsymbol{\nabla} v dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \mathbf{n} \cdot (v\hat{\boldsymbol{\sigma}}) ds - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} v f dx = R_u \quad (\text{A.16})$$

Apply the product rule and divergence theorem for a second time to the third term of eqn. (A.15):

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\tau} \cdot \boldsymbol{\sigma}_A dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \mathbf{n} \cdot (\hat{u}\boldsymbol{\tau}) ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\nabla} \cdot (u_A \boldsymbol{\tau}) dx \\ - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\tau} \cdot \boldsymbol{\nabla} u_A dx = R_\sigma \end{aligned} \quad (\text{A.17})$$

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\tau} \cdot \boldsymbol{\sigma}_A dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \mathbf{n} \cdot (\hat{u}\boldsymbol{\tau}) ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \mathbf{n} \cdot (u_A \boldsymbol{\tau}) ds \\ - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\tau} \cdot \boldsymbol{\nabla} u_A dx = R_\sigma \end{aligned} \quad (\text{A.18})$$

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\tau} \cdot \boldsymbol{\sigma}_A dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \mathbf{n} \cdot (\hat{u} - u_A) \boldsymbol{\tau} ds - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\tau} \cdot \boldsymbol{\nabla} u_A dx = R_\sigma \quad (\text{A.19})$$

Now define  $\boldsymbol{\tau} = \boldsymbol{\nabla} v$  and substitute eqn. (A.19) into eqn. (A.16):

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\boldsymbol{\nabla} u_A \cdot \boldsymbol{\nabla} v - v f) dx + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \mathbf{n} \cdot ((\hat{u} - u_A) \boldsymbol{\nabla} v - v \hat{\boldsymbol{\sigma}}) ds = R \quad (\text{A.20})$$

where  $R = R_u + R_\sigma$ .

This gives a single unified weak form. However, it is generally easier to perform numerical integration over mesh element facets instead of mesh element edges. Each mesh

element edge  $\partial\mathcal{K}$  has two facets,  $\mathcal{F}^+$  and  $\mathcal{F}^-$ , on the interior and exterior respectively of the mesh element under consideration. Both facets have a unit normal,  $\mathbf{n}^+$  and  $\mathbf{n}^-$ , and these normals are equal and opposite to each other  $\mathbf{n}^+ = -\mathbf{n}^-$ . An integral over a mesh element edge can then be rewritten as follows:

$$\int_{\partial\mathcal{K}} \mathbf{n} \cdot q\boldsymbol{\phi} ds = \int_{\mathcal{F}^+} \mathbf{n}^+ \cdot q^+ \boldsymbol{\phi}^+ ds + \int_{\mathcal{F}^-} \mathbf{n}^- \cdot q^- \boldsymbol{\phi}^- ds \quad (\text{A.21})$$

$$= \int_{\mathcal{F}} (\mathbf{n}^+ \cdot q^+ \boldsymbol{\phi}^+ + \mathbf{n}^- \cdot q^- \boldsymbol{\phi}^-) ds \quad (\text{A.22})$$

where superscript (+) and (-) indicate the values of the scalar  $q$  and vector  $\boldsymbol{\phi}$  on the interior and exterior of the mesh element respectively. However, this is only true of mesh element edges in the interior of the simulation domain ( $\mathcal{F}_I$ ); mesh element edges on the boundary of the simulation domain ( $\mathcal{F}_B$ ) only have one single facet. Accounting for this, sums over all mesh elements of integrals over mesh element edges can be rewritten as follows:

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \mathbf{n} \cdot q\boldsymbol{\phi} ds = \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} (\mathbf{n}^+ \cdot q^+ \boldsymbol{\phi}^+ + \mathbf{n}^- \cdot q^- \boldsymbol{\phi}^-) ds + \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \mathbf{n} \cdot q\boldsymbol{\phi} ds \quad (\text{A.23})$$

To simplify notation and follow standard convention define average  $\{\cdot\}$  and jump  $[[\cdot]]$  operators:

$$\{q\} := \begin{cases} \frac{1}{2}(q^+ + q^-) & \text{interior edges} \\ q & \text{exterior edges} \end{cases} \quad (\text{A.24})$$

$$[[q]] := \begin{cases} q^+ - q^- & \text{interior edges} \\ q & \text{exterior edges} \end{cases} \quad (\text{A.25})$$

Consider the following combination of these operators:

$$\begin{aligned} [[q\mathbf{n}]] \cdot \{\phi\} + \{q\}[[\phi \cdot \mathbf{n}]] &= (q^+ \mathbf{n}^+ + q^- \mathbf{n}^-) \cdot \frac{1}{2} (\phi^+ + \phi^-) \\ &\quad + \frac{1}{2} (q^+ + q^-) (\phi^+ \cdot \mathbf{n}^+ + \phi^- \cdot \mathbf{n}^-) \end{aligned} \quad (\text{A.26})$$

$$\begin{aligned} &= \frac{1}{2} \left( 2q^+ \mathbf{n}^+ \cdot \phi^+ + q^+ \mathbf{n}^+ \cdot \phi^- + q^+ \mathbf{n}^- \cdot \phi^- \right. \\ &\quad \left. + q^- \mathbf{n}^- \cdot \phi^+ + q^- \mathbf{n}^+ \cdot \phi^+ + 2q^- \mathbf{n}^- \cdot \phi^- \right) \end{aligned} \quad (\text{A.27})$$

$$\begin{aligned} &= \frac{1}{2} \left( 2q^+ \mathbf{n}^+ \cdot \phi^+ + q^+ \mathbf{n}^+ \cdot \phi^- - q^+ \mathbf{n}^+ \cdot \phi^- \right. \\ &\quad \left. + q^- \mathbf{n}^- \cdot \phi^+ - q^- \mathbf{n}^- \cdot \phi^+ + 2q^- \mathbf{n}^- \cdot \phi^- \right) \end{aligned} \quad (\text{A.28})$$

$$= q^+ \mathbf{n}^+ \cdot \phi^+ + q^- \mathbf{n}^- \cdot \phi^- \quad (\text{A.29})$$

Now use eqn. (A.29) to rewrite eqn. (A.23) in terms of the average and jump operators:

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \mathbf{n} \cdot q \phi \, ds = \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} (\mathbf{n}^+ \cdot q^+ \phi^+ + \mathbf{n}^- \cdot q^- \phi^-) \, ds + \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \mathbf{n} \cdot q \phi \, ds \quad (\text{A.30})$$

$$= \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[q\mathbf{n}]] \cdot \{\phi\} + \{q\}[[\phi \cdot \mathbf{n}]] \right) \, ds + \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \mathbf{n} \cdot q \phi \, ds \quad (\text{A.31})$$

Insert eqn. (A.31) into eqn. (A.20) to put the weak form in standard notation:

$$\begin{aligned} &\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nabla u_A \cdot \nabla v - v f) \, dx \\ &+ \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[(\hat{u} - u_A) \mathbf{n}]] \cdot \{\nabla v\} + \{\hat{u} - u_A\} [[\nabla v \cdot \mathbf{n}]] \right) \, ds \\ &- \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[v \mathbf{n}]] \cdot \{\hat{\sigma}\} - \{v\} [[\hat{\sigma} \cdot \mathbf{n}]] \right) \, ds \\ &+ \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \mathbf{n} \cdot \left( (\hat{u} - u_A) \nabla v - v \hat{\sigma} \right) \, ds = R \end{aligned} \quad (\text{A.32})$$

Now the values of  $\hat{u}$  and  $\hat{\sigma}$  must be defined. This thesis uses the interior penalty

discontinuous Galerkin method [35]:

$$\hat{u} = \begin{cases} \{u_A\} & \text{on } \mathcal{F}_I \\ g & \text{on } \mathcal{F}_D \\ u_A & \text{on } \mathcal{F}_N \\ u_A & \text{on } \mathcal{F}_R \end{cases} \quad (\text{A.33})$$

$$\hat{\boldsymbol{\sigma}} = \begin{cases} \{\nabla u_A\} - \alpha[[u_A \mathbf{n}]] & \text{on } \mathcal{F}_I \\ \nabla u_A - \alpha(u_A - g) \mathbf{n} & \text{on } \mathcal{F}_D \\ h \mathbf{n} & \text{on } \mathcal{F}_N \\ -r(u_A - q) \mathbf{n} & \text{on } \mathcal{F}_R \end{cases} \quad (\text{A.34})$$

where  $\alpha$  is a penalty coefficient typically taken as  $\alpha = \frac{10n^2}{h}$  for a polynomial interpolant order  $n$  and mesh element size  $h$ .

Substitute these values into eqn. (A.32):

$$\begin{aligned} & \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nabla u_A \cdot \nabla v - v f) dx \\ & + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[(\{u_A\} - u_A) \mathbf{n}]] \cdot \{\nabla v\} + \{\{u_A\} - u_A\} [[\nabla v \cdot \mathbf{n}]] \right) ds \\ & - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[v \mathbf{n}]] \cdot \{\{\nabla u_A\} - \alpha[[u_A \mathbf{n}]]\} - \{v\} [[\{\nabla u_A\} \cdot \mathbf{n} - \alpha[[u_A]]]] \right) ds \\ & + \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \mathbf{n} \cdot \left( (g - u_A) \nabla v - v (\nabla u_A - \alpha(u_A - g) \mathbf{n}) \right) ds \\ & + \sum_{\mathcal{F} \in \mathcal{F}_N} \int_{\mathcal{F}} \mathbf{n} \cdot \left( (u_A - u_A) \nabla v - v h \mathbf{n} \right) ds \\ & + \sum_{\mathcal{F} \in \mathcal{F}_R} \int_{\mathcal{F}} \mathbf{n} \cdot \left( (u_A - u_A) \nabla v + v r (u_A - q) \mathbf{n} \right) ds = R \end{aligned} \quad (\text{A.35})$$

Now simplify by noting that the average of an average or jump is just that average or jump

and the jump of an average or jump is zero:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nabla u_A \cdot \nabla v - v f) dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[u_A \mathbf{n}]] \cdot \{\nabla v\} + [[v \mathbf{n}]] \cdot \{\nabla u_A\} - \alpha [[u_A]] [[v]] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \left( \mathbf{n} \cdot (u_A - g) \nabla v + \mathbf{n} \cdot v \nabla u_A - \alpha (u_A - g) v \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_N} \int_{\mathcal{F}} v h ds + \sum_{\mathcal{F} \in \mathcal{F}_R} \int_{\mathcal{F}} v r (u_A - q) ds = R
\end{aligned} \tag{A.36}$$

Eqn. (A.36) is the final weak form for the discontinuous Galerkin formulation of the Poisson equation and is in the correct form to be passed into a finite element method solver.

## A.2 The Stokes Equations

Consider a domain  $\Omega$  with boundary  $\Gamma$  meshed by a triangulation  $\mathcal{T}$  of mesh elements  $\mathcal{K}$ . The Stokes equations are defined as follows with possible Dirichlet and normal stress boundary conditions on mesh boundary regions  $\Gamma_D$  and  $\Gamma_S$ :

$$-\nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega \tag{A.37}$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \tag{A.38}$$

$$\mathbf{u} = \mathbf{g} \quad \text{on } \Gamma_D \tag{A.39}$$

$$(-\nu \nabla \mathbf{u} + p \mathbb{I}) \cdot \mathbf{n} = \mathbf{h} \quad \text{on } \Gamma_S \tag{A.40}$$

where  $\mathbf{f}$  is some body force and  $\nu$  is the kinematic viscosity.

Initially only consider the conservation of momentum equation eqn. (A.37). Define  $\boldsymbol{\sigma}$  to be the gradient of  $\mathbf{u}$  and use it to reduce this second order partial differential equation into a redundant system of first order partial differential equations:

$$\boldsymbol{\sigma} = \nabla \mathbf{u} \tag{A.41}$$

$$-\nu \nabla \cdot \boldsymbol{\sigma} + \nabla p = \mathbf{f} \tag{A.42}$$

Insert trial functions  $\boldsymbol{\sigma}_A$ ,  $\mathbf{u}_A$ , and  $p_A$  to form residuals and average with weighting functions  $\boldsymbol{\tau}$  for eqn. (A.41) and  $\mathbf{v}$  for eqn. (A.42):

$$\int_{\Omega} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \int_{\Omega} \nabla \mathbf{u}_A : \boldsymbol{\tau} \, dx = R_{\sigma} \quad (\text{A.43})$$

$$\int_{\Omega} \nu \mathbf{v} \cdot \nabla \cdot \boldsymbol{\sigma}_A \, dx - \int_{\Omega} \mathbf{v} \cdot \nabla p_A \, dx + \int_{\Omega} \mathbf{v} \cdot \mathbf{f} \, dx = R_u \quad (\text{A.44})$$

Apply the product rule and then the divergence theorem to eqn. (A.43):

$$\int_{\Omega} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \int_{\Omega} \nabla \cdot (\mathbf{u}_A \cdot \boldsymbol{\tau}) \, dx + \int_{\Omega} \mathbf{u}_A \cdot (\nabla \cdot \boldsymbol{\tau}) \, dx = R_{\sigma} \quad (\text{A.45})$$

$$\int_{\Omega} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \int_{\Gamma} \mathbf{u}_A \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds + \int_{\Omega} \mathbf{u}_A \cdot (\nabla \cdot \boldsymbol{\tau}) \, dx = R_{\sigma} \quad (\text{A.46})$$

and to eqn. (A.44):

$$- \int_{\Omega} \left( \nu \boldsymbol{\sigma}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx + \int_{\Omega} \nabla \cdot \left( \nu \mathbf{v} \cdot \boldsymbol{\sigma}_A - p_A \mathbf{v} \right) dx = R_u \quad (\text{A.47})$$

$$- \int_{\Omega} \left( \nu \boldsymbol{\sigma}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx + \int_{\Gamma} \mathbf{v} \cdot \left( \nu \boldsymbol{\sigma}_A - p_A \mathbb{I} \right) \cdot \mathbf{n} \, ds = R_u \quad (\text{A.48})$$

For convenience, define  $\mathbf{H} = \nu \boldsymbol{\sigma}_A - p_A \mathbb{I}$  and use it to rewrite eqn. (A.48):

$$- \int_{\Omega} \left( \nu \boldsymbol{\sigma}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx + \int_{\Gamma} \mathbf{v} \cdot \mathbf{H} \cdot \mathbf{n} \, ds = R_u \quad (\text{A.49})$$

Substituting eqn. (A.46) into eqn. (A.49) produces the standard finite element method weak form. Using the discontinuous Galerkin method instead, the trial functions  $\mathbf{u}_A$ ,  $\boldsymbol{\sigma}_A$ , and  $p_A$  are allowed to be discontinuous across mesh element boundaries. Integrals over the entire simulation domain  $\Omega$  become the sum of integrals over individual mesh elements  $\mathcal{K}$ . Furthermore, integrals over the simulation domain boundary  $\Gamma$  now include all mesh element boundaries  $\partial \mathcal{K}$ . The degeneracy of the trial functions on mesh element boundaries poses a problem for boundary integral terms as they may evaluate to different values within each of the two mesh elements that share any given mesh element boundary. To resolve this, new variables are defined which are single-valued on mesh element boundaries. These variables,  $\hat{\mathbf{H}}$  and  $\hat{\boldsymbol{\sigma}}$ , will be referred to as facet values. They are functions of  $\mathbf{u}_A$ ,  $\boldsymbol{\sigma}_A$ , and  $p_A$  and their specific definitions will be given later in this section. Accounting for this,

eqn. (A.46) and eqn. (A.49) become as follows:

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \hat{\mathbf{u}} \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{u}_A \cdot (\nabla \cdot \boldsymbol{\tau}) \, dx = R_\sigma \quad (\text{A.50})$$

$$- \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \boldsymbol{\sigma}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \mathbf{v} \cdot \hat{\mathbf{H}} \cdot \mathbf{n} \, ds = R_u \quad (\text{A.51})$$

Apply the product rule and divergence theorem for a second time to the third term of eqn. (A.50):

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \hat{\mathbf{u}} \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nabla \cdot (\mathbf{u}_A \cdot \boldsymbol{\tau}) \, dx \\ - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nabla \mathbf{u}_A : \boldsymbol{\tau} \, dx = R_\sigma \end{aligned} \quad (\text{A.52})$$

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \hat{\mathbf{u}} \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \mathbf{u}_A \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds \\ - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nabla \mathbf{u}_A : \boldsymbol{\tau} \, dx = R_\sigma \end{aligned} \quad (\text{A.53})$$

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} (\hat{\mathbf{u}} - \mathbf{u}_A) \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nabla \mathbf{u}_A : \boldsymbol{\tau} \, dx = R_\sigma \quad (\text{A.54})$$

Now define  $\boldsymbol{\tau} = \nabla \mathbf{v}$  and substitute eqn. (A.54) into eqn. (A.51):

$$\begin{aligned} - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\ + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \left( \mathbf{v} \cdot \hat{\mathbf{H}} \cdot \mathbf{n} - \nu (\hat{\mathbf{u}} - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds = R \end{aligned} \quad (\text{A.55})$$

where  $R_m = R_u + R_\sigma$ .

This gives a single unified weak form. However, it is generally easier to perform numerical integration over mesh element facets instead of mesh element edges. Each mesh element edge  $\partial \mathcal{K}$  has two facets,  $\mathcal{F}^+$  and  $\mathcal{F}^-$ , on the interior and exterior respectively of the mesh element under consideration. Both facets have a unit normal,  $\mathbf{n}^+$  and  $\mathbf{n}^-$ , and these normals are equal and opposite to each other  $\mathbf{n}^+ = -\mathbf{n}^-$ . An integral over a mesh

element edge can then be rewritten as follows:

$$\int_{\partial\kappa} \boldsymbol{\phi} \cdot \boldsymbol{\gamma} \cdot \mathbf{n} \, ds = \int_{\mathcal{F}^+} \boldsymbol{\phi}^+ \cdot \boldsymbol{\gamma}^+ \cdot \mathbf{n}^+ \, ds + \int_{\mathcal{F}^-} \boldsymbol{\phi}^- \cdot \boldsymbol{\gamma}^- \cdot \mathbf{n}^- \, ds \quad (\text{A.56})$$

$$= \int_{\mathcal{F}} (\boldsymbol{\phi}^+ \cdot \boldsymbol{\gamma}^+ \cdot \mathbf{n}^+ + \boldsymbol{\phi}^- \cdot \boldsymbol{\gamma}^- \cdot \mathbf{n}^-) \, ds \quad (\text{A.57})$$

where superscript (+) and (-) indicate the values of the vector  $\boldsymbol{\phi}$  and second order tensor  $\boldsymbol{\gamma}$  on the interior and exterior of the mesh element respectively. However, this is only true of mesh element edges in the interior of the simulation domain ( $\mathcal{F}_I$ ); mesh element edges on the boundary of the simulation domain ( $\mathcal{F}_B$ ) only have one single facet. Accounting for this, sums over all mesh elements of integrals over mesh element edges can be rewritten as follows:

$$\begin{aligned} \sum_{\kappa \in \mathcal{T}} \int_{\partial\kappa} \boldsymbol{\phi} \cdot \boldsymbol{\gamma} \cdot \mathbf{n} \, ds &= \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} (\boldsymbol{\phi}^+ \cdot \boldsymbol{\gamma}^+ \cdot \mathbf{n}^+ + \boldsymbol{\phi}^- \cdot \boldsymbol{\gamma}^- \cdot \mathbf{n}^-) \, ds \\ &+ \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \boldsymbol{\phi} \cdot \boldsymbol{\gamma} \cdot \mathbf{n} \, ds \end{aligned} \quad (\text{A.58})$$

To simplify notation and follow standard convention define average  $\{\cdot\}$  and jump  $[[\cdot]]$  operators for general tensor variables:

$$\{\boldsymbol{\phi}\} := \begin{cases} \frac{1}{2}(\boldsymbol{\phi}^+ + \boldsymbol{\phi}^-) & \text{interior edges} \\ \boldsymbol{\phi} & \text{exterior edges} \end{cases} \quad (\text{A.59})$$

$$[[\boldsymbol{\phi}]] := \begin{cases} \boldsymbol{\phi}^+ - \boldsymbol{\phi}^- & \text{interior edges} \\ \boldsymbol{\phi} & \text{exterior edges} \end{cases} \quad (\text{A.60})$$



Consider the following combination of these operators:

$$\begin{aligned}
& \{\phi\} \cdot [[\gamma \cdot \mathbf{n}]] + [[\phi \mathbf{n}]] : \{\gamma\} \\
&= \frac{1}{2} (\phi^+ + \phi^-) \cdot (\gamma^+ \cdot \mathbf{n}^+ + \gamma^- \cdot \mathbf{n}^-) + (\phi^+ \mathbf{n}^+ + \phi^- \mathbf{n}^-) : \frac{1}{2} (\gamma^+ + \gamma^-) \\
&= \frac{1}{2} \left( \phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^+ \cdot \gamma^- \cdot \mathbf{n}^- + \phi^- \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^- \cdot \mathbf{n}^- \right. \\
&\quad \left. + \phi^+ \mathbf{n}^+ : \gamma^+ + \phi^+ \mathbf{n}^+ : \gamma^- + \phi^- \mathbf{n}^- : \gamma^+ + \phi^- \mathbf{n}^- : \gamma^- \right) \\
&= \frac{1}{2} \left( \phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^+ \cdot \gamma^- \cdot \mathbf{n}^- + \phi^- \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^- \cdot \mathbf{n}^- \right. \\
&\quad \left. + \phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^+ \cdot \gamma^- \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^+ \cdot \mathbf{n}^- + \phi^- \cdot \gamma^- \cdot \mathbf{n}^- \right) \\
&= \frac{1}{2} \left( 2\phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^+ \cdot \gamma^- \cdot \mathbf{n}^- - \phi^+ \cdot \gamma^- \cdot \mathbf{n}^- \right. \\
&\quad \left. + \phi^- \cdot \gamma^+ \cdot \mathbf{n}^+ - \phi^- \cdot \gamma^+ \cdot \mathbf{n}^+ + 2\phi^- \cdot \gamma^- \cdot \mathbf{n}^- \right) \\
&= \phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^- \cdot \mathbf{n}^- \tag{A.61}
\end{aligned}$$

where  $\phi$  is a vector and  $\gamma$  is a second order tensor. Note that the following identity was used to rewrite some of the terms:

$$\phi \cdot \gamma \cdot \mathbf{n} = \phi \mathbf{n} : \gamma \tag{A.62}$$

Now use eqn. (A.61) to rewrite eqn. (A.58) in terms of the average and jump operators:

$$\begin{aligned}
\sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \phi \cdot \gamma \cdot \mathbf{n} \, ds &= \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} (\phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^- \cdot \mathbf{n}^-) \, ds \\
&+ \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \phi \cdot \gamma \cdot \mathbf{n} \, ds \tag{A.63}
\end{aligned}$$

$$\begin{aligned}
&= \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\phi\} \cdot [[\gamma \cdot \mathbf{n}]] + [[\phi \mathbf{n}]] : \{\gamma\} \right) \, ds \\
&+ \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \phi \cdot \gamma \cdot \mathbf{n} \, ds \tag{A.64}
\end{aligned}$$

Insert eqn. (A.64) into eqn. (A.55) to put the weak form in standard notation:

$$\begin{aligned}
& - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\mathbf{v}\} \cdot [[\hat{\mathbf{H}} \cdot \mathbf{n}]] + [[\mathbf{v} \mathbf{n}]] : \{\hat{\mathbf{H}}\} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\nu (\hat{\mathbf{u}} - \mathbf{u}_A)\} \cdot [[\nabla \mathbf{v} \cdot \mathbf{n}]] + [[\nu (\hat{\mathbf{u}} - \mathbf{u}_A) \mathbf{n}]] : \{\nabla \mathbf{v}\} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \left( \mathbf{v} \cdot \hat{\mathbf{H}} \cdot \mathbf{n} - \nu (\hat{\mathbf{u}} - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds = R_m
\end{aligned} \tag{A.65}$$

Now the values of  $\hat{\mathbf{u}}$  and  $\hat{\mathbf{H}}$  must be defined. This thesis uses the interior penalty discontinuous Galerkin method [35]:

$$\hat{\mathbf{u}} = \begin{cases} \{\mathbf{u}_A\} & \text{on } \mathcal{F}_I \\ \mathbf{g} & \text{on } \mathcal{F}_D \\ \mathbf{u}_A & \text{on } \mathcal{F}_S \end{cases} \tag{A.66}$$

$$\hat{\mathbf{H}} = \begin{cases} \nu \{\nabla \mathbf{u}_A\} - \nu \alpha [[\mathbf{u}_A \mathbf{n}]] & \text{on } \mathcal{F}_I \\ \nu \nabla \mathbf{u}_A - \nu \alpha (\mathbf{u}_A - \mathbf{g}) \mathbf{n} & \text{on } \mathcal{F}_D \\ -\nu \nabla \mathbf{u}_A + p_A \mathbb{I} & \text{on } \mathcal{F}_S \end{cases} \tag{A.67}$$

where  $\alpha$  is a penalty coefficient typically taken as  $\alpha = \frac{10n^2}{h}$  for a polynomial interpolant order  $n$  and mesh element size  $h$ .

Substitute these values into eqn. (A.65):

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\mathbf{v}\} \cdot \left[ \nu \{\nabla \mathbf{u}_A\} \cdot \mathbf{n} - \nu \alpha [\mathbf{u}_A \mathbf{n}] \cdot \mathbf{n} \right] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[\mathbf{v} \mathbf{n}]] : \left\{ \nu \{\nabla \mathbf{u}_A\} - \nu \alpha [\mathbf{u}_A \mathbf{n}] \right\} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \left\{ \nu (\{\mathbf{u}_A\} - \mathbf{u}_A) \right\} \cdot [[\nabla \mathbf{v} \cdot \mathbf{n}]] + [[\nu (\{\mathbf{u}_A\} - \mathbf{u}_A) \mathbf{n}]] : \{\nabla \mathbf{v}\} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \left( \mathbf{v} \cdot (\nu \nabla \mathbf{u}_A - \nu \alpha (\mathbf{u}_A - \mathbf{g}) \mathbf{n}) \cdot \mathbf{n} - \nu (\mathbf{g} - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \left( \mathbf{v} \cdot (-\nu \nabla \mathbf{u}_A + p_A \mathbb{I}) \cdot \mathbf{n} - \nu (\mathbf{u}_A - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds = R_m \quad (\text{A.68})
\end{aligned}$$

Now simplify by noting that the average of an average or jump is just that average or jump and the jump of an average or jump is zero and using identity eqn. (A.62):

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v} \mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v} \mathbf{n}]] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v} \mathbf{n} : \nabla \mathbf{u}_A + (\mathbf{u}_A - \mathbf{g}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \left( -\nu \nabla \mathbf{u}_A + p_A \mathbb{I} \right) \cdot \mathbf{n} ds = R_m \quad (\text{A.69})
\end{aligned}$$

Finally, notice that the term in the stress boundary condition integral can be replaced by

the value of said stress boundary condition:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A + (\mathbf{u}_A - \mathbf{g}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \mathbf{h} ds = R_m
\end{aligned} \tag{A.70}$$

Eqn. (A.70) is the final weak form for the conservation of momentum equation.

Now the conservation of mass equation eqn. (A.38) can be considered. Insert trial function  $\mathbf{u}_A$  to form the residual and average with weighting function  $q$ :

$$\int_{\Omega} q (\nabla \cdot \mathbf{u}_A) dx = R_c \tag{A.71}$$

Then allow the trial function and weighting function to be discontinuous:

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} q (\nabla \cdot \mathbf{u}_A) dx = R_c \tag{A.72}$$

Finally, sum together eqn. (A.70) and eqn. (A.72) noting that  $R = R_m + R_c$ :

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A + (\mathbf{u}_A - \mathbf{g}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \mathbf{h} ds = R
\end{aligned} \tag{A.73}$$

Eqn. (A.73) is the final weak form for the discontinuous Galerkin formulation of the Stokes equations and is in the correct form to be passed into a finite element method solver.

However, in some cases it is desirable to include a stabilization term  $\epsilon p_A q$ , where  $\epsilon \approx 1 \times 10^{-10}$ , giving the following formulation:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \epsilon p_A q - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v} \mathbf{n}]] : \{ \nabla \mathbf{u}_A \} + [[\mathbf{u}_A \mathbf{n}]] : \{ \nabla \mathbf{v} \} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v} \mathbf{n}]] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v} \mathbf{n} : \nabla \mathbf{u}_A + (\mathbf{u}_A - \mathbf{g}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \mathbf{h} ds = R
\end{aligned} \tag{A.74}$$

### A.3 The Incompressible Navier-Stokes Equations

Consider a domain  $\Omega$  with boundary  $\Gamma$  meshed by a triangulation  $\mathcal{T}$  of mesh elements  $\mathcal{K}$ . The incompressible Navier–Stokes equations are defined as follows with possible Dirichlet and normal stress boundary conditions on mesh boundary regions  $\Gamma_D$  and  $\Gamma_S$ :

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u} \mathbf{u}) - \nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega \tag{A.75}$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \tag{A.76}$$

$$\mathbf{u}(t = 0) = \mathbf{u}_0 \quad \text{in } \Omega \tag{A.77}$$

$$p(t = 0) = p_0 \quad \text{in } \Omega \tag{A.78}$$

$$\mathbf{u} = \mathbf{g} \quad \text{on } \Gamma_D \tag{A.79}$$

$$(\mathbf{u} \mathbf{u} - \nu \nabla \mathbf{u} + p \mathbb{I}) \cdot \mathbf{n} - \max(\mathbf{u} \cdot \mathbf{n}, 0) \mathbf{u} = \mathbf{h} \quad \text{on } \Gamma_S \tag{A.80}$$

where  $\mathbf{f}$  is some body force and  $\nu$  is the kinematic viscosity. The formulation of the normal stress boundary condition to additionally enforce no-backflow comes from [90].

Initially only consider the conservation of momentum equation eqn. (A.75). Define  $\boldsymbol{\sigma}$  to be the gradient of  $\mathbf{u}$  and use it to reduce this second order partial differential equation into a redundant system of first order partial differential equations:

$$\boldsymbol{\sigma} = \nabla \mathbf{u} \tag{A.81}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u} \mathbf{u}) - \nu \nabla \cdot \boldsymbol{\sigma} + \nabla p = \mathbf{f} \tag{A.82}$$

Insert trial functions  $\boldsymbol{\sigma}_A$ ,  $\mathbf{u}_A$ , and  $p_A$  to form residuals and average with weighting functions  $\boldsymbol{\tau}$  for eqn. (A.81) and  $\mathbf{v}$  for eqn. (A.82):

$$\int_{\Omega} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \int_{\Omega} \nabla \mathbf{u}_A : \boldsymbol{\tau} \, dx = R_{\sigma} \quad (\text{A.83})$$

$$\int_{\Omega} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} + \mathbf{v} \cdot \nabla \cdot (\mathbf{u}_A \mathbf{u}_A) - \nu \mathbf{v} \cdot \nabla \cdot \boldsymbol{\sigma}_A + \mathbf{v} \cdot \nabla p_A \right) dx - \int_{\Omega} \mathbf{v} \cdot \mathbf{f} \, dx = R_u \quad (\text{A.84})$$

Apply the product rule and then the divergence theorem to eqn. (A.83):

$$\int_{\Omega} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \int_{\Omega} \nabla \cdot (\mathbf{u}_A \cdot \boldsymbol{\tau}) \, dx + \int_{\Omega} \mathbf{u}_A \cdot (\nabla \cdot \boldsymbol{\tau}) \, dx = R_{\sigma} \quad (\text{A.85})$$

$$\int_{\Omega} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \int_{\Gamma} \mathbf{u}_A \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds + \int_{\Omega} \mathbf{u}_A \cdot (\nabla \cdot \boldsymbol{\tau}) \, dx = R_{\sigma} \quad (\text{A.86})$$

and to eqn. (A.84):

$$\begin{aligned} \int_{\Omega} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{u}_A : \nabla \mathbf{v} + \nu \boldsymbol{\sigma}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\ + \int_{\Omega} \nabla \cdot \left( \mathbf{v} \cdot \mathbf{u}_A \mathbf{u}_A - \nu \mathbf{v} \cdot \boldsymbol{\sigma}_A + p_A \mathbf{v} \right) dx = R_u \end{aligned} \quad (\text{A.87})$$

$$\begin{aligned} \int_{\Omega} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{u}_A : \nabla \mathbf{v} + \nu \boldsymbol{\sigma}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\ + \int_{\Gamma} \left( \mathbf{v} \cdot \mathbf{u}_A \mathbf{u}_A - \nu \mathbf{v} \cdot \boldsymbol{\sigma}_A + p_A \mathbf{v} \right) \cdot \mathbf{n} \, ds = R_u \end{aligned} \quad (\text{A.88})$$

For convenience, define  $\mathbf{H} = \mathbf{u}_A \mathbf{u}_A - \nu \boldsymbol{\sigma}_A + p_A \mathbb{I}$  and use it to rewrite eqn. (A.88):

$$\begin{aligned} \int_{\Omega} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{u}_A : \nabla \mathbf{v} + \nu \boldsymbol{\sigma}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\ + \int_{\Gamma} \mathbf{v} \cdot \mathbf{H} \cdot \mathbf{n} \, ds = R_u \end{aligned} \quad (\text{A.89})$$

Substituting eqn. (A.86) into eqn. (A.89) produces the standard finite element method weak form. Using the discontinuous Galerkin method instead, the trial functions  $\mathbf{u}_A$ ,  $\boldsymbol{\sigma}_A$ , and  $p_A$  are allowed to be discontinuous across mesh element boundaries. Integrals over the entire simulation domain  $\Omega$  become the sum of integrals over individual mesh elements  $\mathcal{K}$ . Furthermore, integrals over the simulation domain boundary  $\Gamma$  now include all mesh element boundaries  $\partial \mathcal{K}$ . The degeneracy of the trial functions on mesh element boundaries

poses a problem for boundary integral terms as they may evaluate to different values within each of the two mesh elements that share any given mesh element boundary. To resolve this, new variables are defined which are single-valued on mesh element boundaries. These variables,  $\hat{\mathbf{H}}$  and  $\hat{\boldsymbol{\sigma}}$ , will be referred to as facet values. They are functions of  $\mathbf{u}_A$ ,  $\boldsymbol{\sigma}_A$ , and  $p_A$  and their specific definitions will be given later in this section. Accounting for this, eqn. (A.86) and eqn. (A.89) become as follows:

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \hat{\mathbf{u}} \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{u}_A \cdot (\nabla \cdot \boldsymbol{\tau}) \, dx &= R_\sigma \quad (\text{A.90}) \\ \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{u}_A : \nabla \mathbf{v} + \nu \boldsymbol{\sigma}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\ + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \mathbf{v} \cdot \hat{\mathbf{H}} \cdot \mathbf{n} \, ds &= R_u \quad (\text{A.91}) \end{aligned}$$

Apply the product rule and divergence theorem for a second time to the third term of eqn. (A.90):

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \hat{\mathbf{u}} \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nabla \cdot (\mathbf{u}_A \cdot \boldsymbol{\tau}) \, dx \\ - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nabla \mathbf{u}_A : \boldsymbol{\tau} \, dx &= R_\sigma \quad (\text{A.92}) \end{aligned}$$

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \hat{\mathbf{u}} \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \mathbf{u}_A \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds \\ - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nabla \mathbf{u}_A : \boldsymbol{\tau} \, dx &= R_\sigma \quad (\text{A.93}) \end{aligned}$$

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \boldsymbol{\sigma}_A : \boldsymbol{\tau} \, dx - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} (\hat{\mathbf{u}} - \mathbf{u}_A) \cdot \boldsymbol{\tau} \cdot \mathbf{n} \, ds - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nabla \mathbf{u}_A : \boldsymbol{\tau} \, dx = R_\sigma \quad (\text{A.94})$$

Now define  $\boldsymbol{\tau} = \nabla \mathbf{v}$  and substitute eqn. (A.94) into eqn. (A.91):

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{u}_A : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\ + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \left( \mathbf{v} \cdot \hat{\mathbf{H}} \cdot \mathbf{n} - \nu (\hat{\mathbf{u}} - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds = R_m \quad (\text{A.95}) \end{aligned}$$

where  $R_m = R_u + R_\sigma$ .

This gives a single unified weak form. However, it is generally easier to perform numerical integration over mesh element facets instead of mesh element edges. Each mesh element edge  $\partial\mathcal{K}$  has two facets,  $\mathcal{F}^+$  and  $\mathcal{F}^-$ , on the interior and exterior respectively of the mesh element under consideration. Both facets have a unit normal,  $\mathbf{n}^+$  and  $\mathbf{n}^-$ , and these normals are equal and opposite to each other  $\mathbf{n}^+ = -\mathbf{n}^-$ . An integral over a mesh element edge can then be rewritten as follows:

$$\int_{\partial\mathcal{K}} \boldsymbol{\phi} \cdot \boldsymbol{\gamma} \cdot \mathbf{n} \, ds = \int_{\mathcal{F}^+} \boldsymbol{\phi}^+ \cdot \boldsymbol{\gamma}^+ \cdot \mathbf{n}^+ \, ds + \int_{\mathcal{F}^-} \boldsymbol{\phi}^- \cdot \boldsymbol{\gamma}^- \cdot \mathbf{n}^- \, ds \quad (\text{A.96})$$

$$= \int_{\mathcal{F}} (\boldsymbol{\phi}^+ \cdot \boldsymbol{\gamma}^+ \cdot \mathbf{n}^+ + \boldsymbol{\phi}^- \cdot \boldsymbol{\gamma}^- \cdot \mathbf{n}^-) \, ds \quad (\text{A.97})$$

where superscript (+) and (-) indicate the values of the vector  $\boldsymbol{\phi}$  and second order tensor  $\boldsymbol{\gamma}$  on the interior and exterior of the mesh element respectively. However, this is only true of mesh element edges in the interior of the simulation domain ( $\mathcal{F}_I$ ); mesh element edges on the boundary of the simulation domain ( $\mathcal{F}_B$ ) only have one single facet. Accounting for this, sums over all mesh elements of integrals over mesh element edges can be rewritten as follows:

$$\begin{aligned} \sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial\mathcal{K}} \boldsymbol{\phi} \cdot \boldsymbol{\gamma} \cdot \mathbf{n} \, ds &= \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} (\boldsymbol{\phi}^+ \cdot \boldsymbol{\gamma}^+ \cdot \mathbf{n}^+ + \boldsymbol{\phi}^- \cdot \boldsymbol{\gamma}^- \cdot \mathbf{n}^-) \, ds \\ &+ \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \boldsymbol{\phi} \cdot \boldsymbol{\gamma} \cdot \mathbf{n} \, ds \end{aligned} \quad (\text{A.98})$$

To simplify notation and follow standard convention define average  $\{\cdot\}$  and jump  $[[\cdot]]$  operators for general tensor variables:

$$\{\boldsymbol{\phi}\} := \begin{cases} \frac{1}{2}(\boldsymbol{\phi}^+ + \boldsymbol{\phi}^-) & \text{interior edges} \\ \boldsymbol{\phi} & \text{exterior edges} \end{cases} \quad (\text{A.99})$$

$$[[\boldsymbol{\phi}]] := \begin{cases} \boldsymbol{\phi}^+ - \boldsymbol{\phi}^- & \text{interior edges} \\ \boldsymbol{\phi} & \text{exterior edges} \end{cases} \quad (\text{A.100})$$



Consider the following combination of these operators:

$$\begin{aligned}
& \{\phi\} \cdot [[\gamma \cdot \mathbf{n}]] + [[\phi \mathbf{n}]] : \{\gamma\} \\
&= \frac{1}{2} (\phi^+ + \phi^-) \cdot (\gamma^+ \cdot \mathbf{n}^+ + \gamma^- \cdot \mathbf{n}^-) + (\phi^+ \mathbf{n}^+ + \phi^- \mathbf{n}^-) : \frac{1}{2} (\gamma^+ + \gamma^-) \\
&= \frac{1}{2} \left( \phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^+ \cdot \gamma^- \cdot \mathbf{n}^- + \phi^- \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^- \cdot \mathbf{n}^- \right. \\
&\quad \left. + \phi^+ \mathbf{n}^+ : \gamma^+ + \phi^+ \mathbf{n}^+ : \gamma^- + \phi^- \mathbf{n}^- : \gamma^+ + \phi^- \mathbf{n}^- : \gamma^- \right) \\
&= \frac{1}{2} \left( \phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^+ \cdot \gamma^- \cdot \mathbf{n}^- + \phi^- \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^- \cdot \mathbf{n}^- \right. \\
&\quad \left. + \phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^+ \cdot \gamma^- \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^+ \cdot \mathbf{n}^- + \phi^- \cdot \gamma^- \cdot \mathbf{n}^- \right) \\
&= \frac{1}{2} \left( 2\phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^+ \cdot \gamma^- \cdot \mathbf{n}^- - \phi^+ \cdot \gamma^- \cdot \mathbf{n}^- \right. \\
&\quad \left. + \phi^- \cdot \gamma^+ \cdot \mathbf{n}^+ - \phi^- \cdot \gamma^+ \cdot \mathbf{n}^+ + 2\phi^- \cdot \gamma^- \cdot \mathbf{n}^- \right) \\
&= \phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^- \cdot \mathbf{n}^- \tag{A.101}
\end{aligned}$$

where  $\phi$  is a vector and  $\gamma$  is a second order tensor. Note that the following identity was used to rewrite some of the terms:

$$\phi \cdot \gamma \cdot \mathbf{n} = \phi \mathbf{n} : \gamma \tag{A.102}$$

Now use eqn. (A.101) to rewrite eqn. (A.98) in terms of the average and jump operators:

$$\begin{aligned}
\sum_{\mathcal{K} \in \mathcal{T}} \int_{\partial \mathcal{K}} \phi \cdot \gamma \cdot \mathbf{n} \, ds &= \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} (\phi^+ \cdot \gamma^+ \cdot \mathbf{n}^+ + \phi^- \cdot \gamma^- \cdot \mathbf{n}^-) \, ds \\
&+ \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \phi \cdot \gamma \cdot \mathbf{n} \, ds \tag{A.103}
\end{aligned}$$

$$\begin{aligned}
&= \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\phi\} \cdot [[\gamma \cdot \mathbf{n}]] + [[\phi \mathbf{n}]] : \{\gamma\} \right) \, ds \\
&+ \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \phi \cdot \gamma \cdot \mathbf{n} \, ds \tag{A.104}
\end{aligned}$$

Insert eqn. (A.104) into eqn. (A.95) to put the weak form in standard notation:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{u}_A : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\mathbf{v}\} \cdot [[\hat{\mathbf{H}} \cdot \mathbf{n}]] + [[\mathbf{v} \mathbf{n}]] : \{\hat{\mathbf{H}}\} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\nu (\hat{\mathbf{u}} - \mathbf{u}_A)\} \cdot [[\nabla \mathbf{v} \cdot \mathbf{n}]] + [[\nu (\hat{\mathbf{u}} - \mathbf{u}_A) \mathbf{n}]] : \{\nabla \mathbf{v}\} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \left( \mathbf{v} \cdot \hat{\mathbf{H}} \cdot \mathbf{n} - \nu (\hat{\mathbf{u}} - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds = R_m
\end{aligned} \tag{A.105}$$

Eqn. (A.105) is nonlinear and thus challenging to solve completely implicitly. One possible solution is to substitute a known velocity field  $\mathbf{w}$  for one of the velocities in the nonlinear convection term:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\mathbf{v}\} \cdot [[\hat{\mathbf{H}} \cdot \mathbf{n}]] + [[\mathbf{v} \mathbf{n}]] : \{\hat{\mathbf{H}}\} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\nu (\hat{\mathbf{u}} - \mathbf{u}_A)\} \cdot [[\nabla \mathbf{v} \cdot \mathbf{n}]] + [[\nu (\hat{\mathbf{u}} - \mathbf{u}_A) \mathbf{n}]] : \{\nabla \mathbf{v}\} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \left( \mathbf{v} \cdot \hat{\mathbf{H}} \cdot \mathbf{n} - \nu (\hat{\mathbf{u}} - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds = R_m
\end{aligned} \tag{A.106}$$

Combining eqn. (A.106) with a conservation of mass equation gives the linear Oseen equations. The value of  $\mathbf{w}$  can come from the solution of a previous time step or be obtained through Picard iterations. An alternative is to treat the convection term explicitly and solve all remaining terms implicitly. This is known as an IMEX (implicit-explicit) scheme [91] and will be given in Appendix A.3.1.

Now the values of  $\hat{\mathbf{u}}$  and  $\hat{\mathbf{H}}$  must be defined. This thesis uses the interior penalty discontinuous Galerkin method [35]. Note that  $\hat{\mathbf{H}}$  is just the summation of the facet values for the convection term, viscous term, and pressure term  $\hat{\mathbf{H}} = \hat{\mathbf{H}}_c + \hat{\mathbf{H}}_v + \hat{\mathbf{H}}_p$ .

Thus, for clarity, the facet value for the convection term is given separately:

$$\hat{\mathbf{u}} = \begin{cases} \{\mathbf{u}_A\} & \text{on } \mathcal{F}_I \\ \mathbf{g} & \text{on } \mathcal{F}_D \\ \mathbf{u}_A & \text{on } \mathcal{F}_S \end{cases} \quad (\text{A.107})$$

$$\hat{\mathbf{H}}_c = \begin{cases} \{\mathbf{u}_A\} (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A^+ - \mathbf{u}_A^-) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} & \text{on } \mathcal{F}_I \\ \frac{1}{2} (\mathbf{u}_A + \mathbf{g}) (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A - \mathbf{g}) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} & \text{on } \mathcal{F}_D \\ \mathbf{u}_A \mathbf{w} & \text{on } \mathcal{F}_S \end{cases} \quad (\text{A.108})$$

$$\hat{\mathbf{H}}_v + \hat{\mathbf{H}}_p = \begin{cases} -\nu \{\nabla \mathbf{u}_A\} + \nu \alpha [[\mathbf{u}_A \mathbf{n}]] & \text{on } \mathcal{F}_I \\ -\nu \nabla \mathbf{u}_A + \nu \alpha (\mathbf{u}_A - \mathbf{g}) \mathbf{n} & \text{on } \mathcal{F}_D \\ -\nu \nabla \mathbf{u}_A + p_A \mathbb{I} & \text{on } \mathcal{F}_S \end{cases} \quad (\text{A.109})$$

where  $\alpha$  is a penalty coefficient typically taken as  $\alpha = \frac{10n^2}{h}$  for a polynomial interpolant order  $n$  and mesh element size  $h$ .

Substitute these values into eqn. (A.106):

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\mathbf{v}\} \cdot [\{\{\mathbf{u}_A\}(\mathbf{w} \cdot \mathbf{n}) + \frac{1}{2}(\mathbf{u}_A^+ - \mathbf{u}_A^-)|\mathbf{w} \cdot \mathbf{n}|\}] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\mathbf{v}\} \cdot [\nu \{\nabla \mathbf{u}_A\} \cdot \mathbf{n} - \nu \alpha [\mathbf{u}_A \mathbf{n}] \cdot \mathbf{n}] \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[\mathbf{v} \mathbf{n}]] : \{\{\mathbf{u}_A\}(\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2}(\mathbf{u}_A^+ - \mathbf{u}_A^-)|\mathbf{w} \cdot \mathbf{n}|\mathbf{n}\} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[\mathbf{v} \mathbf{n}]] : \{\nu \{\nabla \mathbf{u}_A\} - \nu \alpha [\mathbf{u}_A \mathbf{n}]\} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\nu(\{\mathbf{u}_A\} - \mathbf{u}_A)\} \cdot [[\nabla \mathbf{v} \cdot \mathbf{n}]] + [[\nu(\{\mathbf{u}_A\} - \mathbf{u}_A) \mathbf{n}]] : \{\nabla \mathbf{v}\} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \mathbf{v} \cdot \left( \frac{1}{2}(\mathbf{u}_A + \mathbf{g})(\mathbf{w} \cdot \mathbf{n}) + \frac{1}{2}(\mathbf{u}_A - \mathbf{g})|\mathbf{w} \cdot \mathbf{n}| \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \mathbf{v} \cdot \left( \nu \nabla \mathbf{u}_A - \nu \alpha (\mathbf{u}_A - \mathbf{g}) \mathbf{n} \cdot \mathbf{n} - \nu (\mathbf{g} - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \left( \mathbf{v} \cdot (\mathbf{u}_A \mathbf{w} - \nu \nabla \mathbf{u}_A + p_A \mathbb{I}) \cdot \mathbf{n} - \nu (\mathbf{u}_A - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds = R_m \quad (\text{A.110})
\end{aligned}$$

Now simplify by noting that the average of an average or jump is just that average or jump

and the jump of an average or jump is zero and using identity eqn. (A.102):

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [[\mathbf{v}\mathbf{n}]] : \left( \{\mathbf{u}_A\} (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A^+ - \mathbf{u}_A^-) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \mathbf{v}\mathbf{n} : \left( \frac{1}{2} (\mathbf{u}_A + \mathbf{g}) (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A - \mathbf{g}) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A + \mathbf{u}_A \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \left( \mathbf{u}_A \mathbf{w} - \nu \nabla \mathbf{u}_A + p_A \mathbb{I} \right) \cdot \mathbf{n} ds = R_m \tag{A.111}
\end{aligned}$$

Finally, notice that the term in the stress boundary condition integral can be replaced by the value of said stress boundary condition:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [[\mathbf{v}\mathbf{n}]] : \left( \{\mathbf{u}_A\} (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A^+ - \mathbf{u}_A^-) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \mathbf{v}\mathbf{n} : \left( \frac{1}{2} (\mathbf{u}_A + \mathbf{g}) (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A - \mathbf{g}) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A + \mathbf{u}_A \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \left( \mathbf{h} + \max(\mathbf{w} \cdot \mathbf{n}, 0) \mathbf{u}_A \right) ds = R_m \tag{A.112}
\end{aligned}$$

Eqn. (A.112) is the final weak form for the conservation of momentum equation.

Now the conservation of mass equation eqn. (A.76) can be considered. Insert trial function  $\mathbf{u}_A$  to form the residual and average with weighting function  $q$ :

$$\int_{\Omega} q(\nabla \cdot \mathbf{u}_A) dx = R_c \quad (\text{A.113})$$

Then allow the trial function and weighting function to be discontinuous:

$$\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} q(\nabla \cdot \mathbf{u}_A) dx = R_c \quad (\text{A.114})$$

Finally, sum together eqn. (A.112) and eqn. (A.114) noting that  $R = R_m + R_c$ :

$$\begin{aligned} & \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q(\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) dx \\ & + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [[\mathbf{v}\mathbf{n}]] : \left( \{\mathbf{u}_A\} (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A^+ - \mathbf{u}_A^-) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\ & - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\ & + \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \mathbf{v}\mathbf{n} : \left( \frac{1}{2} (\mathbf{u}_A + \mathbf{g}) (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A - \mathbf{g}) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\ & - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A + \mathbf{u}_A \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\ & + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \left( \mathbf{h} + \max(\mathbf{w} \cdot \mathbf{n}, 0) \mathbf{u}_A \right) ds = R \end{aligned} \quad (\text{A.115})$$

Eqn. (A.115) is the final weak form for the discontinuous Galerkin formulation of the incompressible Navier–Stokes equations. The time derivative must be discretized but otherwise it is in the correct form to be passed into a finite element method solver. However, in some cases it is desirable to include a stabilization term  $\epsilon p_A q$ , where  $\epsilon \approx 1 \times 10^{-10}$ , giving the

following formulation:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \epsilon p_A q \right) dx \\
& - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \mathbf{f} dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [[\mathbf{v}\mathbf{n}]] : \left( \{\mathbf{u}_A\} (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A^+ - \mathbf{u}_A^-) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \mathbf{v}\mathbf{n} : \left( \frac{1}{2} (\mathbf{u}_A + \mathbf{g}) (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A - \mathbf{g}) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A + \mathbf{u}_A \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \left( \mathbf{h} + \max(\mathbf{w} \cdot \mathbf{n}, 0) \mathbf{u}_A \right) ds = R
\end{aligned} \tag{A.116}$$

### A.3.1 IMEX Time Discretization

The full discontinuous Galerkin formulation of the weak form of the incompressible Navier-Stokes equations is given below prior to defining expressions for the facet values:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{u}_A : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \epsilon p_A q \right) dx \\
& - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \mathbf{f} dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\mathbf{v}\} \cdot [[\hat{\mathbf{H}} \cdot \mathbf{n}]] + [[\mathbf{v}\mathbf{n}]] : \{\hat{\mathbf{H}}\} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( \{\nu (\hat{\mathbf{u}} - \mathbf{u}_A)\} \cdot [[\nabla \mathbf{v} \cdot \mathbf{n}]] + [[\nu (\hat{\mathbf{u}} - \mathbf{u}_A) \mathbf{n}]] : \{\nabla \mathbf{v}\} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} \left( \mathbf{v} \cdot \hat{\mathbf{H}} \cdot \mathbf{n} - \nu (\hat{\mathbf{u}} - \mathbf{u}_A) \cdot \nabla \mathbf{v} \cdot \mathbf{n} \right) ds = R
\end{aligned} \tag{A.117}$$

The convection term makes eqn. (A.117) nonlinear. The previous section handled this by replacing one of the velocities with a known velocity field,  $\mathbf{w}$ , making eqn. (A.117) linear. Another option is to leave the convection term as is but treat it explicitly in the time discretization scheme. A fully explicit time discretization scheme, such as explicit Euler, could be used. However, the viscous term is generally stiff so explicit schemes require a very small time step to ensure stability. A better option is to couple an explicit treatment of the nonlinear convection term with an implicit treatment of the remaining terms. This is known as an IMEX (implicit-explicit) time discretization scheme and enables the use of large time steps without requiring a nonlinear or iterative solve [91]. In the case of the incompressible Navier-Stokes equations, the terms treated implicitly comprise a Stokes system, so can further take advantage of optimized Stokes solvers and preconditioners [92].

Ascher *et al* [91] describe a variety of IMEX schemes of different orders. For simplicity, this derivation will consider a basic first-order scheme:

$$\frac{\mathbf{u}_A^{n+1} - \mathbf{u}_A^n}{\Delta t} + F^n + G^{n+1} = R \quad (\text{A.118})$$

where  $n$  and  $n + 1$  represent the current and future time steps respectively and  $\Delta t$  is the time step size.  $F^n$  comprises all terms that are to be treated explicitly, evaluated at the current time step, and  $G^{n+1}$  comprises all terms that are to be treated implicitly and whose values at the future time step are to be solved for. Considering eqn. (A.117),  $F^n$  is as follows:

$$F^n = - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{u}_A^n \mathbf{u}_A^n : \nabla \mathbf{v} \, dx \quad (\text{A.119})$$



and  $G^{n+1}$ :

$$\begin{aligned}
G^{n+1} &= \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nu \nabla \mathbf{u}_A^{n+1} : \nabla \mathbf{v} - p_A^{n+1} (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A^{n+1}) - \epsilon p_A^{n+1} q) dx \\
&\quad - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \mathbf{f}^{n+1} dx \\
&\quad + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} (\{\mathbf{v}\} \cdot [[\hat{\mathbf{H}}^{n+1} \cdot \mathbf{n}]] + [[\mathbf{v}\mathbf{n}]] : \{\hat{\mathbf{H}}^{n+1}\}) ds \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \{\nu (\hat{\mathbf{u}}^{n+1} - \mathbf{u}_A^{n+1})\} \cdot [[\nabla \mathbf{v} \cdot \mathbf{n}]] ds \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [[\nu (\hat{\mathbf{u}}^{n+1} - \mathbf{u}_A^{n+1}) \mathbf{n}]] : \{\nabla \mathbf{v}\} ds \\
&\quad + \sum_{\mathcal{F} \in \mathcal{F}_B} \int_{\mathcal{F}} (\mathbf{v} \cdot \hat{\mathbf{H}}^{n+1} \cdot \mathbf{n} - \nu (\hat{\mathbf{u}}^{n+1} - \mathbf{u}_A^{n+1}) \cdot \nabla \mathbf{v} \cdot \mathbf{n}) ds \tag{A.120}
\end{aligned}$$

Now the values of  $\hat{\mathbf{u}}$  and  $\hat{\mathbf{H}}$  can be defined. Note that since the convection term is treated fully explicitly  $\hat{\mathbf{H}}$  only comprises the facet values for the viscous term and pressure term:

$$\hat{\mathbf{u}} = \begin{cases} \{\mathbf{u}_A\} & \text{on } \mathcal{F}_I \\ \mathbf{g} & \text{on } \mathcal{F}_D \\ \mathbf{u}_A & \text{on } \mathcal{F}_S \end{cases} \tag{A.121}$$

$$\hat{\mathbf{H}} = \begin{cases} -\nu \{\nabla \mathbf{u}_A\} + \nu \alpha [[\mathbf{u}_A \mathbf{n}]] & \text{on } \mathcal{F}_I \\ -\nu \nabla \mathbf{u}_A + \nu \alpha (\mathbf{u}_A - \mathbf{g}) \mathbf{n} & \text{on } \mathcal{F}_D \\ -\nu \nabla \mathbf{u}_A + p_A \mathbb{I} & \text{on } \mathcal{F}_S \end{cases} \tag{A.122}$$

where  $\alpha$  is a penalty coefficient typically taken as  $\alpha = \frac{10n^2}{h}$  for a polynomial interpolant order  $n$  and mesh element size  $h$ .

Substitute these values into eqn. (A.120):

$$\begin{aligned}
G^{n+1} &= \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nu \nabla \mathbf{u}_A^{n+1} : \nabla \mathbf{v} - p_A^{n+1} (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A^{n+1}) - \epsilon p_A^{n+1} q) dx \\
&\quad - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \mathbf{f}^{n+1} dx \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \{ \mathbf{v} \} \cdot [ [\nu \{ \nabla \mathbf{u}_A^{n+1} \} \cdot \mathbf{n} - \nu \alpha [ [\mathbf{u}_A^{n+1} \mathbf{n}] ] \cdot \mathbf{n} ] ] ds \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [ [ \mathbf{v} \mathbf{n} ] ] : \{ \nu \{ \nabla \mathbf{u}_A^{n+1} \} - \nu \alpha [ [\mathbf{u}_A^{n+1} \mathbf{n}] ] \} ds \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \{ \nu (\{ \mathbf{u}_A^{n+1} \} - \mathbf{u}_A^{n+1}) \} \cdot [ [ \nabla \mathbf{v} \cdot \mathbf{n} ] ] ds \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [ [ \nu (\{ \mathbf{u}_A^{n+1} \} - \mathbf{u}_A^{n+1}) \mathbf{n} ] ] : \{ \nabla \mathbf{v} \} ds \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} (\mathbf{v} \mathbf{n} : (\nu \nabla \mathbf{u}_A^{n+1} - \nu \alpha (\mathbf{u}_A^{n+1} - \mathbf{g}^{n+1}) \mathbf{n}) - \nu (\mathbf{g}^{n+1} - \mathbf{u}_A^{n+1}) \cdot \nabla \mathbf{v} \cdot \mathbf{n}) ds \\
&\quad + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} (\mathbf{v} \cdot (-\nu \nabla \mathbf{u}_A^{n+1} + p_A^{n+1} \mathbb{I})) \cdot \mathbf{n} - \nu (\mathbf{u}_A^{n+1} - \mathbf{u}_A^{n+1}) \cdot \nabla \mathbf{v} \cdot \mathbf{n}) ds \quad (\text{A.123})
\end{aligned}$$

Now simplify by noting that the average of an average or jump is just that average or jump and the jump of an average or jump is zero and using identity eqn. (A.102):

$$\begin{aligned}
G^{n+1} &= \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nu \nabla \mathbf{u}_A^{n+1} : \nabla \mathbf{v} - p_A^{n+1} (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A^{n+1}) - \epsilon p_A^{n+1} q) dx \\
&\quad - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \mathbf{f}^{n+1} dx \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [ [ \mathbf{v} \mathbf{n} ] ] : \{ \nabla \mathbf{u}_A^{n+1} \} + [ [ \mathbf{u}_A^{n+1} \mathbf{n} ] ] : \{ \nabla \mathbf{v} \} - \alpha [ [ \mathbf{u}_A^{n+1} \mathbf{n} ] ] : [ [ \mathbf{v} \mathbf{n} ] ] \right) ds \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu (\mathbf{v} \mathbf{n} : \nabla \mathbf{u}_A^{n+1} + (\mathbf{u}_A^{n+1} - \mathbf{g}^{n+1}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A^{n+1} - \mathbf{g}^{n+1}) \cdot \mathbf{v}) ds \\
&\quad + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot (-\nu \nabla \mathbf{u}_A^{n+1} + p_A^{n+1} \mathbb{I}) \cdot \mathbf{n} ds \quad (\text{A.124})
\end{aligned}$$

Finally, notice that the term in the stress boundary condition integral can be replaced by the value of said stress boundary condition:

$$\begin{aligned}
G^{n+1} &= \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nu \nabla \mathbf{u}_A^{n+1} : \nabla \mathbf{v} - p_A^{n+1} (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A^{n+1}) - \epsilon p_A^{n+1} q) dx \\
&\quad - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \mathbf{f}^{n+1} dx \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A^{n+1}\} + [[\mathbf{u}_A^{n+1}\mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A^{n+1}\mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\
&\quad - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A^{n+1} + (\mathbf{u}_A^{n+1} - \mathbf{g}^{n+1}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A^{n+1} - \mathbf{g}^{n+1}) \cdot \mathbf{v} \right) ds \\
&\quad + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \mathbf{h} ds \tag{A.125}
\end{aligned}$$

Then the final weak form, including time discretization, is as follows:

$$\begin{aligned}
&\sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \frac{\mathbf{u}_A^{n+1} - \mathbf{u}_A^n}{\Delta t} \\
&- \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{u}_A^n \mathbf{u}_A^n : \nabla \mathbf{v} dx \\
&+ \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nu \nabla \mathbf{u}_A^{n+1} : \nabla \mathbf{v} - p_A^{n+1} (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A^{n+1}) - \epsilon p_A^{n+1} q) dx \\
&- \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \mathbf{f}^{n+1} dx \\
&- \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A^{n+1}\} + [[\mathbf{u}_A^{n+1}\mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A^{n+1}\mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\
&- \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A^{n+1} + (\mathbf{u}_A^{n+1} - \mathbf{g}^{n+1}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A^{n+1} - \mathbf{g}^{n+1}) \cdot \mathbf{v} \right) ds \\
&+ \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \mathbf{h} ds = R \tag{A.126}
\end{aligned}$$

# Appendix B

## Diffuse Interface Formulations of OpenCMP Models

The following models were developed with the help of James Lowman and are based on work by Nguyen *et al* [38].

### B.1 The Poisson Equation

Consider the Poisson equation defined on a complex domain  $\Omega$  with boundaries  $\Gamma_D$ ,  $\Gamma_N$ , and  $\Gamma_R$  corresponding to Dirichlet, Neumann, and Robin boundary conditions respectively:

$$-\nabla^2 u = f \quad \text{in } \Omega \quad (\text{B.1})$$

$$u = g \quad \text{on } \Gamma_D \quad (\text{B.2})$$

$$-\mathbf{n} \cdot \nabla u = h \quad \text{on } \Gamma_N \quad (\text{B.3})$$

$$\mathbf{n} \cdot \nabla u = r(u - q) \quad \text{on } \Gamma_R \quad (\text{B.4})$$

where  $f$  is some source function.

The standard finite element method weak form is derived in Appendix A to be eqn. (A.14) and is repeated below with boundary conditions inserted:

$$\int_{\Omega} \nabla u_A \cdot \nabla v \, dx + \int_{\Gamma_N} v h \, ds - \int_{\Gamma_R} v r (u_A - q) \, ds - \int_{\Omega} v f \, dx = R \quad (\text{B.5})$$

Now enclose the complex domain  $\Omega$  in a simple domain  $\kappa$  and discretize it with a mesh that does not necessarily conform to the boundaries of  $\Omega$ . Define a phase field  $\phi$  which takes the value one at any mesh elements within  $\Omega$  and zero at any mesh elements outside of  $\Omega$ , and varies smoothly from zero to one across the boundary  $\Gamma$ . The following identities can be used to replace volume integrals over  $\Omega$  and surface integrals over  $\Gamma$  with volume integrals over  $\kappa$  [38]:

$$\int_{\Omega} A dx = \int_{\kappa} AH dx \approx \int_{\kappa} A\phi dx \quad (\text{B.6})$$

$$\int_{\Gamma} B ds = \int_{\kappa} \delta_{\Gamma} B dx \approx \int_{\kappa} B|\nabla\phi| dx \quad (\text{B.7})$$

$$\mathbf{n} \approx \frac{-\nabla\phi}{|\nabla\phi|} \quad (\text{B.8})$$

where  $H$  is the Heaviside function and  $\delta_{\Gamma}$  is the Dirac delta function, which are approximated by  $\phi$  and  $|\nabla\phi|$  respectively.

Substituting these identities into eqn. (B.5) gives the diffuse interface formulation of the finite element weak form for the Poisson equation with Neumann and Robin boundary conditions:

$$\int_{\kappa} \nabla u_A \cdot \nabla v\phi dx + \int_{\kappa} vh|\nabla\phi|\phi_N dx - \int_{\kappa} vr(u_A - q)|\nabla\phi|\phi_R dx - \int_{\kappa} vf\phi dx = R \quad (\text{B.9})$$

Note that  $\phi_N$  and  $\phi_R$  are masks to mark the portions of the phase field corresponding to Neumann and Robin boundary conditions respectively. For more details see work by Monte *et al* [8].

Dirichlet boundary conditions are imposed by directly setting the value of the trial function at the relevant mesh nodes. However, this is not possible with the diffuse interface method as the mesh does not necessarily conform to the boundary of the complex domain. Instead, the Nitsche method [93] can be used to weakly impose Dirichlet boundary conditions in the region of the complex boundary. This requires the following additions to the weak form:

$$\begin{aligned} & \int_{\kappa} \nabla u_A \cdot \nabla v\phi dx + \int_{\kappa} vh|\nabla\phi|\phi_N dx - \int_{\kappa} vr(u_A - q)|\nabla\phi|\phi_R dx - \int_{\kappa} vf\phi dx \\ & + \int_{\kappa} (u_A - g) \nabla\phi \cdot \nabla v\phi_D dx + \int_{\kappa} v\nabla\phi \cdot \nabla u_A\phi_D dx \\ & + \beta \int_{\kappa} v(u_A - g)|\nabla\phi|\phi_D dx = R \end{aligned} \quad (\text{B.10})$$

where  $\beta$  is a penalty parameter in this work taken to be  $\beta = \frac{10n^2}{h}$  for a polynomial interpolant order  $n$  and mesh element size  $h$ . Eqn. (B.10) is the diffuse interface formulation of the standard finite element method weak form for the Poisson equation.

The discontinuous Galerkin formulation can be obtained by the same means. Begin with the weak form for a complex domain-conforming mesh (eqn. (A.36)):

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nabla u_A \cdot \nabla v - v f) dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[u_A \mathbf{n}]] \cdot \{\nabla v\} + [[v \mathbf{n}]] \cdot \{\nabla u_A\} - \alpha [[u_A]] [[v]] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \left( \mathbf{n} \cdot (u_A - g) \nabla v + \mathbf{n} \cdot v \nabla u_A - \alpha (u_A - g) v \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_N} \int_{\mathcal{F}} v h ds + \sum_{\mathcal{F} \in \mathcal{F}_R} \int_{\mathcal{F}} v r (u_A - q) ds = R
\end{aligned} \tag{B.11}$$

Redefine the triangulation  $\mathcal{T}$  to be for a nonconforming mesh. Then use eqns. (B.6) to (B.8) to convert all volume and surface integrals on the conforming mesh into volume integrals on the nonconforming mesh:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} (\nabla u_A \cdot \nabla v - v f) \phi dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \left( [[u_A \mathbf{n}]] \cdot \{\nabla v\} + [[v \mathbf{n}]] \cdot \{\nabla u_A\} - \alpha [[u_A]] [[v]] \right) \phi ds \\
& + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( (u_A - g) \nabla \phi \cdot \nabla v + v \nabla \phi \cdot \nabla u_A + \beta (u_A - g) v |\nabla \phi| \right) \phi_D dx \\
& + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} v h |\nabla \phi| \phi_N dx \\
& - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} v r (u_A - q) |\nabla \phi| \phi_R dx = R
\end{aligned} \tag{B.12}$$

Eqn. (B.12) is the diffuse interface formulation of the discontinuous Galerkin method weak form for the Poisson equation. Note that integrals over boundary facets have been replaced by volume integrals due to the lack of complex boundary-conforming mesh elements. However, the integrals over interior facets used to penalize discontinuities in the final solution

field remain facet integrals as the nonconforming mesh still contains interior facets. Also note that the discontinuous Galerkin terms used to weakly impose Dirichlet boundary conditions become the Nitsche method when boundary facet integrals are transformed into volume integrals. The discontinuous Galerkin penalty parameter  $\alpha$  is generally equal to the Nitsche method penalty parameter  $\beta$ .

## B.2 The Stokes Equations

Consider the Stokes equations defined on a complex domain  $\Omega$  with boundaries  $\Gamma_D$  and  $\Gamma_S$  corresponding to Dirichlet and normal stress boundary conditions respectively:

$$-\nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega \quad (\text{B.13})$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \quad (\text{B.14})$$

$$\mathbf{u} = \mathbf{g} \quad \text{on } \Gamma_D \quad (\text{B.15})$$

$$(-\nu \nabla \mathbf{u} + p \mathbb{I}) \cdot \mathbf{n} = \mathbf{h} \quad \text{on } \Gamma_S \quad (\text{B.16})$$

where  $\mathbf{f}$  is some body force and  $\nu$  is the kinematic viscosity.

The standard finite element method weak form is derived in Appendix A to be the sum of eqn. (A.49) and eqn. (A.71) and is repeated below with boundary conditions inserted:

$$\int_{\Omega} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) dx + \int_{\Gamma_S} \mathbf{v} \cdot \mathbf{h} ds = R \quad (\text{B.17})$$

Now enclose the complex domain  $\Omega$  in a simple domain  $\kappa$  and discretize it with a mesh that does not necessarily conform to the boundaries of  $\Omega$ . Define a phase field  $\phi$  which takes the value one at any mesh elements within  $\Omega$  and zero at any mesh elements outside of  $\Omega$ , and varies smoothly from zero to one across the boundary  $\Gamma$ . The following identities can be used to replace volume integrals over  $\Omega$  and surface integrals over  $\Gamma$  with volume integrals over  $\kappa$  [38]:

$$\int_{\Omega} A dx = \int_{\kappa} AH dx \approx \int_{\kappa} A\phi dx \quad (\text{B.18})$$

$$\int_{\Gamma} B ds = \int_{\kappa} \delta_{\Gamma} B dx \approx \int_{\kappa} B |\nabla \phi| dx \quad (\text{B.19})$$

$$\mathbf{n} \approx \frac{-\nabla \phi}{|\nabla \phi|} \quad (\text{B.20})$$

where  $H$  is the Heaviside function and  $\delta_\Gamma$  is the Dirac delta function, which are approximated by  $\phi$  and  $|\nabla\phi|$  respectively.

Substituting these identities into eqn. (B.17) gives the diffuse interface formulation of the finite element weak form for the Stokes equations with normal stress boundary conditions:

$$\int_{\kappa} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) \phi \, dx + \int_{\kappa} \mathbf{v} \cdot \mathbf{h} |\nabla \phi| \phi_S \, dx = R \quad (\text{B.21})$$

Note that  $\phi_S$  is a mask that marks the portions of the phase field corresponding to normal stress boundary conditions. For more details see work by Monte *et al* [8].

Dirichlet boundary conditions are imposed by directly setting the value of the trial function at the relevant mesh nodes. However, this is not possible with the diffuse interface method as the mesh does not necessarily conform to the boundary of the complex domain. Instead, the Nitsche method [93] can be used to weakly impose Dirichlet boundary conditions in the region of the boundary. This requires the following additions to the weak form:

$$\begin{aligned} & \int_{\kappa} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) \phi \, dx + \int_{\kappa} \mathbf{v} \cdot \mathbf{h} |\nabla \phi| \phi_S \, dx \\ & + \int_{\kappa} (\mathbf{u}_A - \mathbf{g}) \cdot \nabla \mathbf{v} \cdot \nabla \phi \phi_D \, dx + \int_{\kappa} \mathbf{v} \cdot \nabla \mathbf{u}_A \cdot \nabla \phi \phi_D \, dx \\ & + \beta \int_{\kappa} \mathbf{v} \cdot (\mathbf{u}_A - \mathbf{g}) |\nabla \phi| \phi_D \, dx = R \end{aligned} \quad (\text{B.22})$$

where  $\beta$  is a penalty parameter in this work taken to be  $\beta = \frac{10n^2}{h}$  for a polynomial interpolant order  $n$  and mesh element size  $h$ . Eqn. (B.22) is the diffuse interface formulation of the standard finite element method weak form for the Stokes equations.

The discontinuous Galerkin formulation can be obtained by the same means. Begin



with the weak form for a complex domain-conforming mesh (eqn. (A.73)):

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v}\mathbf{n} : \nabla \mathbf{u}_A + (\mathbf{u}_A - \mathbf{g}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \mathbf{h} ds = R
\end{aligned} \tag{B.23}$$

Redefine the triangulation  $\mathcal{T}$  to be for a nonconforming mesh. Then use eqns. (B.18) to (B.20) to convert all volume and surface integrals on the conforming mesh into volume integrals on the nonconforming mesh:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) \phi dx \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v}\mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v}\mathbf{n}]] \right) \phi ds \\
& + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nu \left( \mathbf{v} \nabla \phi : \nabla \mathbf{u}_A + (\mathbf{u}_A - \mathbf{g}) \nabla \phi : \nabla \mathbf{v} - \beta (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} |\nabla \phi| \right) \phi_D dx \\
& - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \mathbf{h} |\nabla \phi| \phi_S dx = R
\end{aligned} \tag{B.24}$$

Eqn. (B.24) is the diffuse interface formulation of the discontinuous Galerkin method weak form for the Stokes equations. Note that integrals over boundary facets have been replaced by volume integrals due to the lack of complex boundary-conforming mesh elements. However, the integrals over interior facets used to penalize discontinuities in the final solution field remain facet integrals as the nonconforming mesh still contains interior facets. Also note that the discontinuous Galerkin terms used to weakly impose Dirichlet boundary conditions become the Nitsche method when boundary facet integrals are transformed into volume integrals. The discontinuous Galerkin penalty parameter  $\alpha$  is generally equal to the Nitsche method penalty parameter  $\beta$ .

### B.3 The Incompressible Navier-Stokes Equations

Consider the incompressible Navier-Stokes equations defined on a complex domain  $\Omega$  with boundaries  $\Gamma_D$  and  $\Gamma_S$  corresponding to Dirichlet and normal stress boundary conditions respectively:

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) - \nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega \quad (\text{B.25})$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \quad (\text{B.26})$$

$$\mathbf{u}(t=0) = \mathbf{u}_0 \quad \text{in } \Omega \quad (\text{B.27})$$

$$p(t=0) = p_0 \quad \text{in } \Omega \quad (\text{B.28})$$

$$\mathbf{u} = \mathbf{g} \quad \text{on } \Gamma_D \quad (\text{B.29})$$

$$(\mathbf{u}\mathbf{u} - \nu \nabla \mathbf{u} + p\mathbb{I}) \cdot \mathbf{n} - \max(\mathbf{u} \cdot \mathbf{n}, 0) \mathbf{u} = \mathbf{h} \quad \text{on } \Gamma_S \quad (\text{B.30})$$

where  $\mathbf{f}$  is some body force and  $\nu$  is the kinematic viscosity.

The standard finite element method weak form is derived in Appendix A to be the sum of eqn. (A.89) and eqn. (A.113) and is repeated below with boundary conditions inserted:

$$\begin{aligned} & \int_{\Omega} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) dx \\ & + \int_{\Gamma_S} \mathbf{v} \cdot \left( \mathbf{h} + \max(\mathbf{w} \cdot \mathbf{n}, 0) \mathbf{u}_A \right) ds = R \end{aligned} \quad (\text{B.31})$$

Eqn. (B.31) uses Oseen-style linearization to handle the nonlinear convection term. However, the following modifications for the diffuse interface method would apply equally to an IMEX-style linearization.

Now enclose the complex domain  $\Omega$  in a simple domain  $\kappa$  and discretize it with a mesh that does not necessarily conform to the boundaries of  $\Omega$ . Define a phase field  $\phi$  which takes the value one at any mesh elements within  $\Omega$  and zero at any mesh elements outside of  $\Omega$ , and varies smoothly from zero to one across the boundary  $\Gamma$ . The following identities can be used to replace volume integrals over  $\Omega$  and surface integrals over  $\Gamma$  with volume integrals over  $\kappa$  [38]:

$$\int_{\Omega} A dx = \int_{\kappa} AH dx \approx \int_{\kappa} A\phi dx \quad (\text{B.32})$$

$$\int_{\Gamma} B ds = \int_{\kappa} \delta_{\Gamma} B dx \approx \int_{\kappa} B |\nabla \phi| dx \quad (\text{B.33})$$

$$\mathbf{n} \approx \frac{-\nabla \phi}{|\nabla \phi|} \quad (\text{B.34})$$

where  $H$  is the Heaviside function and  $\delta_\Gamma$  is the Dirac delta function, which are approximated by  $\phi$  and  $|\nabla\phi|$  respectively.

Substituting these identities into eqn. (B.31) gives the diffuse interface formulation of the finite element weak form for the incompressible Navier-Stokes equations with normal stress boundary conditions:

$$\begin{aligned} & \int_\kappa \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) \phi \, dx \\ & + \int_\kappa \mathbf{v} \cdot \left( \mathbf{h} + \max(\mathbf{w} \cdot \nabla \phi, 0) \mathbf{u}_A \right) |\nabla \phi| \phi_S \, dx = R \end{aligned} \quad (\text{B.35})$$

Note that  $\phi_S$  is a mask that marks the portions of the phase field corresponding to normal stress boundary conditions. For more details see work by Monte *et al* [8].

Dirichlet boundary conditions are imposed by directly setting the value of the trial function at the relevant mesh nodes. However, this is not possible with the diffuse interface method as the mesh does not necessarily conform to the boundary of the complex domain. Instead, the Nitsche method [93] can be used to weakly impose Dirichlet boundary conditions in the region of the boundary. This requires the following additions to the weak form:

$$\begin{aligned} & \int_\kappa \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) \phi \, dx \\ & + \int_\kappa \mathbf{v} \cdot \left( \mathbf{h} + \max(\mathbf{w} \cdot \nabla \phi, 0) \mathbf{u}_A \right) |\nabla \phi| \phi_S \, dx \\ & + \int_\kappa (\mathbf{u}_A - \mathbf{g}) \cdot \nabla \mathbf{v} \cdot \nabla \phi \phi_D \, dx + \int_\kappa \mathbf{v} \cdot \nabla \mathbf{u}_A \cdot \nabla \phi \phi_D \, dx \\ & + \beta \int_\kappa \mathbf{v} \cdot (\mathbf{u}_A - \mathbf{g}) |\nabla \phi| \phi_D \, dx = R \end{aligned} \quad (\text{B.36})$$

where  $\beta$  is a penalty parameter in this work taken to be  $\beta = \frac{10n^2}{h}$  for a polynomial interpolant order  $n$  and mesh element size  $h$ . Eqn. (B.36) is the diffuse interface formulation of the standard finite element method weak form for the incompressible Navier-Stokes equations.

The discontinuous Galerkin formulation can be obtained by the same means. Begin

with the weak form for a complex domain-conforming mesh (eqn. (A.115)):

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [[\mathbf{v} \mathbf{n}]] : \left( \{\mathbf{u}_A\} (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A^+ - \mathbf{u}_A^-) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v} \mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v} \mathbf{n}]] \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \mathbf{v} \mathbf{n} : \left( \frac{1}{2} (\mathbf{u}_A + \mathbf{g}) (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A - \mathbf{g}) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_D} \int_{\mathcal{F}} \nu \left( \mathbf{v} \mathbf{n} : \nabla \mathbf{u}_A + (\mathbf{u}_A - \mathbf{g}) \mathbf{n} : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} \right) ds \\
& + \sum_{\mathcal{F} \in \mathcal{F}_S} \int_{\mathcal{F}} \mathbf{v} \cdot \left( \mathbf{h} + \max(\mathbf{w} \cdot \mathbf{n}, 0) \mathbf{u}_A \right) ds = R \tag{B.37}
\end{aligned}$$

Redefine the triangulation  $\mathcal{T}$  to be for a nonconforming mesh. Then use eqns. (B.32) to (B.34) to convert all volume and surface integrals on the conforming mesh into volume integrals on the nonconforming mesh:

$$\begin{aligned}
& \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \left( \mathbf{v} \cdot \frac{\partial \mathbf{u}_A}{\partial t} - \mathbf{u}_A \mathbf{w} : \nabla \mathbf{v} + \nu \nabla \mathbf{u}_A : \nabla \mathbf{v} - p_A (\nabla \cdot \mathbf{v}) - q (\nabla \cdot \mathbf{u}_A) - \mathbf{v} \cdot \mathbf{f} \right) \phi dx \\
& + \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} [[\mathbf{v} \mathbf{n}]] : \left( \{\mathbf{u}_A\} (\mathbf{w} \cdot \mathbf{n}) \mathbf{n} + \frac{1}{2} (\mathbf{u}_A^+ - \mathbf{u}_A^-) |\mathbf{w} \cdot \mathbf{n}| \mathbf{n} \right) \phi ds \\
& - \sum_{\mathcal{F} \in \mathcal{F}_I} \int_{\mathcal{F}} \nu \left( [[\mathbf{v} \mathbf{n}]] : \{\nabla \mathbf{u}_A\} + [[\mathbf{u}_A \mathbf{n}]] : \{\nabla \mathbf{v}\} - \alpha [[\mathbf{u}_A \mathbf{n}]] : [[\mathbf{v} \mathbf{n}]] \right) \phi ds \\
& - \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \left( \frac{1}{2} (\mathbf{u}_A + \mathbf{g}) (\mathbf{w} \cdot \nabla \phi) + \frac{1}{2} (\mathbf{u}_A - \mathbf{g}) |\mathbf{w} \cdot \nabla \phi| \right) \phi_D dx \\
& + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \nu \left( \mathbf{v} \nabla \phi : \nabla \mathbf{u}_A + (\mathbf{u}_A - \mathbf{g}) \nabla \phi : \nabla \mathbf{v} - \alpha (\mathbf{u}_A - \mathbf{g}) \cdot \mathbf{v} |\nabla \phi| \right) \phi_D dx \\
& + \sum_{\mathcal{K} \in \mathcal{T}} \int_{\mathcal{K}} \mathbf{v} \cdot \left( \mathbf{h} + \max(\mathbf{w} \cdot \nabla \phi, 0) \mathbf{u}_A \right) |\nabla \phi| \phi_S dx = R \tag{B.38}
\end{aligned}$$

Eqn. (B.38) is the diffuse interface formulation of the discontinuous Galerkin method weak form for the incompressible Navier-Stokes equations. Note that integrals over bound-

ary facets have been replaced by volume integrals due to the lack of complex boundary-conforming mesh elements. However, the integrals over interior facets used to penalize discontinuities in the final solution field remain facet integrals as the nonconforming mesh still contains interior facets. Also note that the discontinuous Galerkin terms used to weakly impose Dirichlet boundary conditions become the Nitsche method, modified to enforce upwinding, when boundary facet integrals are transformed into volume integrals. The discontinuous Galerkin penalty parameter  $\alpha$  is generally equal to the Nitsche method penalty parameter  $\beta$ .