UiO : University of Oslo

Tormod Ravnanger Landet

Discontinuous Galerkin methods for multiphase flow

Thesis submitted for the degree of Philosophiae Doctor

Department of Mathematics
Faculty of Mathematics and Natural Sciences

University of Oslo

2020

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. XYZX

ISSN XYZY-XYZYX

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

Abstract

This thesis describes a finite element method for simulation of free-surface flows, such as ocean waves, using the discontinuous Galerkin method. Free-surface flows where there is a large difference in density between two immiscible fluids will have a large jump in the magnitude of the momentum field at the interface between the two fluids. Such discontinuities create problems for higher-order discretisations, which are more computationally efficient where the solution is smooth, but need careful handling near discontinuities in order to avoid Gibbs oscillations. This thesis shows how slope limiting can be used to stabilise the momentum equation in an exactly mass conserving discontinuous Galerkin finite element discretisation of the Navier–Stokes equation. The stabilised method is tested on a range of well known 2D and 3D free-surface-flow test cases. The results are in good agreement with published experimental results, and also give the expected higher-order convergence rates for smooth solutions.

Preface

I have been incredibly lucky to be able to work on whatever I wanted for more or less the entirety of my scholarship. I want to give my sincerest thanks to the Department of Mathematics at the University of Oslo for the opportunity to have such a free position. There may have been times when I would gladly have traded some of that academic freedom for a position in a team—doing a more collaborative, shared-frustrations type of research—but in the end, maybe the humbling experience of going it alone is an integral part of the PhD experience. What I do know is that I am incredibly proud of what I have accomplished, though I sometimes question how on earth it could take so long.

On the topic of efficiency, it took me six years of working as a wave-loads consultant to finally come to the realisation that the time was right for further studies. I had always said that I would eventually apply for a PhD position, but I think my boss, Olav Rognebakke, may have almost given up on encouraging me to get on with it when I finally sat down and wrote the application letter. It was seeing how much fun and personal growth can be had from working on a PhD project that made me finally want to do it, and I must thank my wife, Ann Kristin Sperrevik, for being so good at leaving most of the frustrations and insecurities inherent in the life of a researcher behind her before coming home every evening—something I have since learned is vital to the mental health of any PhD student. Thank you for showing me the all the good sides of research, and also for acknowledging the less good sides as I discovered them.

I would like to thank all my colleagues at DNV GL for keeping me sharp all those years away from university. Life as a consultant was both enjoyable and rewarding, mostly due to the great team spirit and all the nice people I worked with. To all my UiO colleagues: thank you for the open social and academic atmosphere; from the ionosphere ninjas and down to the planet core deep thinkers, with everything from oceans to Kolmogorov scales in between, thank you! There are so many of you with brilliant ideas and can-do attitudes. I would love to have some alternative time lines to work with all of you and learn more about your academic interests, but at least we got to share the most important part of each day together, the lunch break! Special thanks to Miro and Diako who were there when I started and made our shared office a welcoming and productive place, and Erika, Susanne and Lisa for initiating and helping organise social and sometimes academically relevant activities. Not every group has such a great positive dynamic, and I am grateful that I ended up in the mechanics group, easily provable to be the best group on campus.

Outside the mechanics group I would also like to thank Lucy, Biljana, Terje and the rest of the administration for your support and nice conversations. Sometimes it really helps to walk up and down a few stairs and get away from

research for a few minutes. I would also like to thank the Ocean Modeling Group in the Department of Marine and Coastal Sciences at Rutgers University for welcoming me into their group and providing a space for me to work for six months during the winter of 2018/2019.

Lastly, I would like to thank my supervisors, always full of smiles and interesting ideas. Special thanks to Mikael Mortensen, my main supervisor, who got stuck with me and my free surface problems, which arguably may not be as explosive as combustion or as beautiful as spectral methods. I had the privilege of choosing you—I hope you do not regret too much your friendly explanation of university application procedures to the inexperienced young man who showed up in your office one day and declared that he was ready to start researching.

Tormod Ravnanger Landet
Oslo, Norway, February 2020

Contents

Chapter 1

Introduction

This thesis starts with a short description of the goals and motivations for the work, followed by a brief introduction to available numerical methods and the reasons why this work uses the discontinuous Galerkin finite element method (DG FEM). The selected method is then introduced along with a summary of the stability problems associated with convective operators in low- and high-order discretisations. An introduction to surface-capturing volume-of-fluid (VOF) methods is also included as this method is used in the papers, and the theory fits the theme of convective stability. No novel research on surface capturing is included in this work.

The main part of the thesis consists of three papers describing a numerical method for free-surface flows, and one extended abstract which has been included to highlight all the work that has gone into the code and documentation for the Ocellaris solver, the code that has been developed through the work described herein. After these papers the thesis ends with a conclusion, a summary of the scientific novelty of the work, and recommendations for further work. A few appendices are included to show examples how the Ocellaris solver can be used to study relatively simple 2D flows, and also to provide additional insight into the behaviour of the numerical scheme, such as the inherent numerical diffusion in the DG FEM scheme and the remaining challenges related to using a slope limiter in the viscous shear layer close to a no-slip wall.

1.1 Problem statement

This thesis describes the development of a new two-phase flow solver designed to study air/water free-surface flows in complex domains. The main goals of the numerical method are:

Goal 1 A higher-order numerical method for air/water free-surface flow.

Goal 2 Flow solver stability independently of the free surface treatment.

Goal 3 Exact mass conservation.

Two-phase free-surface flows can mean many things to different people. In this work the focus is on medium-scale phenomena where viscosity plays a minor role, such as ocean waves breaking into structures. Small-scale phenomena, such as contact angles, wetting properties, and even surface tension itself, are disregarded along with large-scale phenomena such as Coriolis forces and density differences due to varying temperature and salinity. Turbulence is also disregarded, though it is of course an important phenomenon at all scales, but not vitally important for the kind of phenomena described herein.

There is nothing fundamentally problematic about adding a turbulence model and surface tension to the described method, but these topics are orthogonal to the convective-stability topics of this thesis and have not been investigated. That does not imply that these topics are easy, or that they are unaffected by the choice of numerical scheme. The discretisation of surface tension and the near-wall behaviour of RANS-type turbulence models are vital for the accuracy of such methods, and these are large topics and highly dependent on the selected discretisation.

The solver targets high-Reynolds-number flow, but this target is not due to a focus on turbulence modelling or turbulent phenomena in this work. Goal 1 implies that the solver should be stable at standard viscosities for air and water, just like it should be stable for a factor-1000 density jump which is what is found at a free surface between water and air. Targeting high-Reynolds-number flow just means that the selected methods should not depend on high-viscosity, near-Stokes-flow, fluid parameters for stability.

1.2 Motivation

Established two-phase flow solvers in this domain (OpenFOAM, StarCCM+, Fluent, FLOW-3D, FINE/Marine, Orca3D, etc.) are all based on low-order finite volume methods with a volume-of-fluid representation of the free surface, a method that goes back to SLIC and SOLA VOF (Hirt and Nichols 1981; Noh and Woodward 1976). These codes use piecewise constant elements, normally with static meshes, possibly with overlapping or gliding meshes to include moving objects. More research-focused codes, such as Gerris and Basilisk, are designed around automatic mesh-refinement methods, and can resolve fine free-surface details in ways that are not generally possible for static-mesh methods without orders of magnitude more computational cost (Popinet 2003; Popinet 2014). This is called h-adaptivity, changing the local cell size parameter h to better resolve details. Another way to improve the solution is to increase the polynomial order of the approximating functions in cells where the solution is under-resolved, called p-adaptivity.

The ultimate computational method—giving the fastest solution for a given allowable error—is adaptive both in mesh cell size and element approximation degree (Babuška and Dorr 1981). Such methods use many small and low-order elements near discontinuities, such as the free surface, and few and large high-order elements in areas where the solution is smooth. Method complexity, assembly speed, matrix reuse, parallelisability, mesh-refinement, CPU cache limitations, efficient preconditioning of the resulting system, and many other limiting factors may in reality lead to a different optimal solution at the current time than the hp-adaptive method. Still, having the option of using higher-order methods—at least in parts of the domain—is important also for free-surface simulations.

Since studying free-surface flows with low order methods is the established state of the art, and h-adaptivity is actively being explored in this domain by

others, this thesis focuses solely on constructing a higher-order method (goal 1 above). Work on higher-order methods for this problem can in time be combined with adaptivity both in space and polynomial order to form a *hp*-adaptive method which should be very competitive, though *hp*-adaptivity is not the focus of this thesis. The aim of the work described in this thesis is therefore not to create a solver that is directly competitive with the state of the art in terms of efficiency, but to create building blocks that can help further the goal of creating *hp*-adaptive methods for free-surface flows which will eventually be much more efficient than what is currently available.

An additional personal motivation is that the developed solver should be fast and scalable enough to at least be able to run some real 3D test cases. This is somewhat in conflict with the above paragraphs, as using higher-order elements everywhere is not the most efficient, but being able to have high-order elements also near the free surface should still be possible. It will not always be feasible, or wanted, to resolve the free surface with a fine low-order mesh in all parts of the domain. Figure 1.1 shows a snapshot from a simulation from Paper III which shows that this is indeed possible.

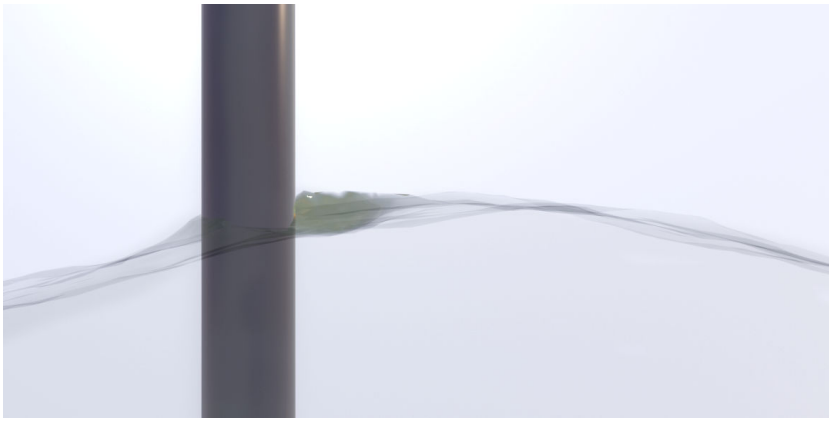


Figure 1.1: Post-processed image from Ocellaris, the free-surface solver that was developed in this work. A wave crest has just passed a vertical rigid cylinder.

The two fluids studied in this work are water and air, and the interface between them is sharp with a large difference in density across the free surface. It is this jump in density combined with the higher-order approximating functions that are at the very core of the included papers. This thesis does not present novel research on free-surface capturing methods, which is an active research field in itself (see section 1.3), but uses a simple VOF method that can represent sharp density transitions to ensure that the method is not stable due to using a smoothed density field, but handles the exact physical surface sharpness (goal 2).

Methods for locating the free-surface in space and time—and hence for dividing the computational domain into air and water sub-domains—are often based on transport of scalar fields, either indicator functions (VOF) or distance

functions (level set). These transport equations requires a divergence free convecting velocity. Any error in mass conservation will not only cause a small error in the velocity field, but will often cause the free surface to move away from its true position, and hence change the geometry of the sub-domains and cause a factor-1000 error in the density field near the free-surface. Mass conservation is hence much more important for free-surface flows than for flows where scalar transport errors are not strongly linked back to the solution of the fluid flow equation. This is the major reason for requiring exact mass conservation (goal 3).

1.3 Method selection

This section describes how the goals from the problem statement above has lead to the selection of the DG FEM solver used in this thesis. Some of the choices below follow directly from the problem statement, for others prior experience and research interests has guided the selection. More details on some of these topics are also given in the literature review that follows.

Complex geometry The intention of the project is to study the interaction between the free-surface and rigid structures present in the wave zone. These structures may be irregular in shape, so a regular mesh can only be used if combined with a method that allows irregular shapes to be embedded without conforming to the mesh, such as the immersed boundary method (Peskin 2002).

Using a regular mesh would have obvious benefits. Cell-based methods can often be more efficient, since the cell-wise matrices are identical for all cells, and often the regular structure can lead to super-convergence (Guillén-González and Tierra 2012). Very-high-order global spectral methods can also be employed, and stabilisation of spectral methods towards shocks such as the jump in density at the free surface do exist, see e.g. Tadmor (1990). Such methods do not easily allow *hp*-adaptivity, which is one reason that a cell-based unstructured mesh was chosen as the method to proceed with.

Higher-order finite difference or finite volume methods are typically also used on regular meshes. A large body or research is dedicated to stable ways to capture shocks and avoid unphysical oscillations, e.g. WENO methods (X.-D. Liu, Osher, and T. Chan 1994; Shu 2003). Such methods have multi-cell stencils which are more complex to integrate with unstructured meshes, but it is possible (Friedrich 1998; Hu and Shu 1999), though non-periodic boundaries are more problematic than for DG methods where multi-cell stencils are not needed.

Anticipating that implementing an immersed boundary method would need more attention than an unstructured mesh method, and pull focus away from the original goals, the more traditional unstructured mesh approach was selected for this work. This choice does not preclude using a very regular mesh in parts of the domain, but we chose to not go further with global spectral methods or multi-cell stencil methods.

Incompressible flow This restriction is important for the selection of a linear-algebra method. Explicit methods can be used for efficient simulation of the compressible Navier–Stokes equations (Hindenlang et al. 2012; Luo et al. 2010), but by requiring a divergence-free velocity field the option to use an explicit solver is no longer possible, and the numerical method must hence use an implicit linear-algebra solver. An exception from this could potentially be methods that use an exactly divergence-free basis for the velocity, allowing the pressure to be eliminated, but constructing a pointwise and globally divergence free basis is not easily done without an elliptic solve, see e.g. Fu (2019). Another option, not taken here, is to include artificial compressibility, see e.g. Bassi et al. (2006) and Bassi et al. (2007).

Exact mass conservation A solver for the incompressible Navier–Stokes equations can be formulated with most discretisation methods, but exact incompressibility requires specific choices. A standard continuous Galerkin FEM can, in general, only become exactly incompressible in the limit of infinitesimally small elements.

Several element types may be used to achieve an exactly divergence-free velocity field. While standard continuous Galerkin methods (CG) will not work, there are many available options that can provide exact mass conservation when used appropriately, e.g. Crouzeix–Raviart, Scott–Vogelius, Raviart–Thomas and Brezzi–Douglas–Marini (Galvin et al. 2012; Kirby, Logg, et al. 2012). Evans (2011) presents a B-spline-based discretisation that provides pointwise divergence free velocities and uses DG methods to discretise non-regular domains by joining together multiple B-spline patches.

It should be noted that the goal is to study incompressible two-phase flow where the velocity field is continuous. Discontinuous methods are used primarily to ensure exact incompressibility, not to resolve shocks. Mimicking the true solution is of course also a possible use of discontinuities in numerical methods, see the discussion on extended finite element methods (XFEM) below.

The discontinuous Galerkin FEM method was selected as the most promising method due to the option to obtain exact incompressibility, ease of higher-order discretisation, and the relative wealth of recent research this method.

High Péclet numbers The cell Péclet number gives the relation between the advection and the diffusion in a computational cell, $\mathrm{Pe}_c = \frac{Uh}{\nu}$, where U is the magnitude of velocity in the cell, h is the characteristic length of the cell and ν is the kinematic viscosity of the fluid. Since the kinematic viscosity is very low for air and water, and similarly small computational cells (in terms of $h \approx \nu$) are not achievable, it is common to have $\mathrm{Pe_c} \gg 1$.

High Péclet numbers require stabilisation in continuous Galerkin methods. A common method is the Streamline Upwind Petrov–Galerkin (SUPG) stabilisation method. Discretisation methods using fluxes, such as finite volume and discontinuous Galerkin methods, can be directly implemented with upwind-

dominated fluxes, which stabilise high-Péclet-number flows (Brooks and Hughes 1982; Hughes 1987).

Free surface location The free surface position can either be explicitly followed by tracking the location of points on the free surface through time (surface tracking), or recreated from a field representation (volume tracking/surface capturing). For an introduction, see Scardovelli and Zaleski (1999). There are also methods such as SPH were the water sub-domain is given by the position of points tracking one of the fluids, though such methods are typically low order (Bøckmann, Shipilova, and Skeie 2012; Monaghan 1994).

Due to the intended use of the solver to study surfaces undergoing large topological changes, such as overturning waves, the interface tracking methods were not investigated. Tracking the geometry of the surface through such changes requires very careful handling of self-intersection and surface break-up due to bubble entrainment or spray. As this aspect was not at the core of the research, the more widely used volume tracking/surface capturing methods were preferred, as they are easy to make robust with regard to topological changes.

Among the volume methods, the two most popular methods are the level-set method method (Osher and Sethian 1988; Sussman, Smereka, and Osher 1994) and the volume-of-fluid (VOF) method (Hirt and Nichols 1981; Noh and Woodward 1976; Youngs 1982). The level set method does not, in its simplest form, preserve mass, while algebraic VOF methods preserve mass by default, but provide worse estimates of surface curvature. Several methods combine the two to provide both mass conservation and better local descriptions of the surface (Olsson and Kreiss 2005; Sussman and Puckett 2000; van der Pijl et al. 2005), or combine level set with particle methods to achieve the same (Enright et al. 2002).

In current state-of-the-art maritime-industry CFD codes the algebraic VOF methods HRIC (Muzaferija et al. 1998) and CICSAM (Ubbink 1997) are almost universally used. The exception is OpenFOAM which uses a similar algebraic VOF method, MULES (H. G. Weller 2008), though geometric VOF methods are also supported (Roenby, Bredmose, and Jasak 2016). The selected method for this work has been HRIC. As the intention has not been to advance the field of interface capturing; using a known, robust, and simple method seemed advantageous compared to implementing a newer and more complex, but potentially more accurate method. See also the related discussed in the section on suggestions for further work (page 142).

Free surface description This topic relates to how the free surface is treated in the Navier–Stokes solver after the location has been determined by a surface capturing method. There are two main choices, one can let the mesh conform to the free surface, or have the free surface embedded in the mesh, cutting through mesh cells at arbitrary positions.

The first option, letting the mesh conform to the free surface, increases the mimetic properties of the method, since the gradient of the pressure has a sharp

change at the free surface, and the density and viscosity fields are discontinuous here. On the other hand, as discussed above, moving the mesh with the motion of the free surface becomes highly non-trivial when the topology of the air and water sub-domains can change significantly by phenomena such as wave over-turning, air-bubble capture, or water spray. Moving the mesh is typically done by the Arbitrary Lagrangian–Eulerian (ALE) method (Hirt, Amsden, and Cook 1974).

An option that allows resolving the discontinuity, while keeping the mesh static, is to use a numerical method that augments the approximating functions in the cells intersected by the free surface—adding local functions that resolve the jump at its exact location. These methods are know as XFEM (Chessa and Belytschko 2003; Groß and Reusken 2007; Heimann et al. 2013; Moës, Dolbow, and Belytschko 1999). Such extended finite element methods may also provide possibilities for resolving jumps without lowering the approximation order near the discontinuity, especially if the interface cells can be cut by higher-order polynomials which resolve the free surface without requiring a large increase in the number of elements.

The method employed in this thesis uses a static mesh and does not try to resolve the free surface discontinuity at the sub-grid level, though that is a very interesting future possibility discussed in Paper IV.

Sharp free surface The jump in the coefficients of the momentum equations, especially the factor-1000 jump in the density between air and water, causes instabilities due to the Gibbs-phenomenon, which causes un-physical wiggles to appear in the velocity field near the free surface. Although the velocities are continuous, the momentum is not, and this discontinuity will quickly cause an instability in the convective operator that blows up and destroys the solution. This is the topic of Paper I, and is also described in the introductory sections 1.6 and 1.7.

This (non-linear) instability can be solved by adding non-linear viscosity to the equations (VonNeumann and Richtmyer 1950), lately made popular by methods such as the entropy-viscosity method (Guermond, Pasquetti, and Popov 2011), which work for both cell-based FEM and global spectral methods. These methods include the artificial viscosity as a part of the linear equations themselves. In general the artificial viscosity methods require equation-specific analysis to determine the required viscosity. A popular alternative is to stabilise the solution process by applying a post-processing filter after the un-stabilised equations have been solved. The filter's job is to remove any un-boundedness, and hence stabilise the time-stepping procedure so that wiggles are not allowed to grow over time. The most common methods are spectral filtering and slope-limiting, see e.g. Michoski et al. (2016) for a comparison of artificial viscosity, spectral filtering and slope limiting.

The method used in this work is based on a element-wise slope-limiting filter. Having the filter be per element greatly simplifies the implementation and using a post-processing filter instead of modifying the weak form of the

governing equation allows decoupling the stabilisation from the equation itself. A Navier–Stokes solver very easily becomes a large piece of tightly coupled code, and when it is possible to enforce some separation of the individual pieces, that is a significant benefit in terms of limiting complexity and being able to try different combinations of methods to find one that suits the individual problem to be solved.

Numerical framework Testing new numerical methods typically requires an actual implementation of the method in computer code to be able to run numerical tests, as it is often hard to analytically show that the method will work, especially for non-linear equations with non-linear instabilities such as the Gibbs phenomenon at the free surface in two-phase Navier–Stokes flow.

One popular way to test a numerical method is to create a proof-of-concept implementation in an easy-to-use programming language like Matlab or Python, and test on small 1D or 2D examples where speed and parallel MPI solution is not crucial. One can of course also code the method in a high performance way, by linking to fast distributed linear algebra libraries such as PETSc (Balay et al. 2018) or Trilinos (Heroux and Willenbring 2003) and implementing the assembly routines in an optimized way (Cython/numba, C, C++, Fortran, Julia etc). FEM frameworks such as DEAL.II (Bangerth, Hartmann, and Kanschat 2007) can do most of this work for you, so that you can code only the matrix assembly inner loops, and let the framework handle the mesh, linear solver and visualisation output from the solver.

FEniCS is an advanced framework which allows the developer of a FEM solver to write the definition of the weak form to be discretised in the Unified Form Language (UFL), which is very close to the mathematical syntax one would write with pen and paper (Alnæs 2012; Alnæs et al. 2014). The developer can then select exactly how this should be converted into code that assembles local element matrices, by selecting everything from element geometry (triangle, quadrilateral etc), dimension (1D/2D/3D), quadrature degree, approximating polynomials, etc. The FEniCS Form Compiler (FFC) takes the mathematical UFL description and automatically generates optimized element-assembly code, without the developer having to spend time chasing off-by-one and plus/minus sign errors in dense numerical code (Kirby and Logg 2006; Logg, K. B. Ølgaard, et al. 2012; K. Ølgaard, Logg, and G. Wells 2008; K. B. Ølgaard and G. N. Wells 2010). FEniCS, through the dolfin library, also integrates with mesh partitioners, MPI libraries, PETSc solvers, Paraview XDMF output for visualisation and provides many other conveniences (Logg, Mardal, and G. Wells 2012). In summary, FEniCS allows fast iteration which allows testing different weak forms and numerical approaches to solve a given problem without having to be stuck writing and debugging optimised assembly and quadrature code every time a change is made.

The selected method This selected method is using element-local Lagrange-polynomial approximating functions in a discontinuous Galerkin setting with a sparse implicit linear-algebra solver. The reason for picking a DG method follows

from the support for higher-order elements on unstructured meshes, support for exact incompressibility and stability at high Péclet numbers. As described above the free surface capturing method is a simple algebraic VOF method, HRIC. A literature review of similar methods is presented in the following section.

The choice of Lagrange polynomials is partly due to the fact that they are well supported in FEniCS, and also that they are easy to work with for custom post-processing (e.g. slope limiting) and visualisation. The Ocellaris solver used in this work (www.ocellaris.org) did itself end up as a small framework for testing different parts of the method, but it is fully dependent on FEniCS for the underlying functionality that is used to discretise and solve the governing equations.

1.4 Literature review

The following sections give a brief history of the development of numerical methods for solving the incompressible Navier–Stokes equations with a focus on free-surface flows and discontinuous Galerkin finite element methods, though other topics are mentioned where considered relevant. A section on selected recent publications on these topics is included at the end.

1.4.1 Free-surface flow

Numerical solution of the viscous free-surface flows started with the marker-and-cell (MAC) method by Harlow and Welch (1965), an Eulerian method based on finite differences for the field solvers and marker particles to track the location of the fluid and hence the free surface. The domain was divided by the free surface into fluid and void. Explicit boundary condition where applied at the free surface to account for the lack of a lighter fluid on top. The MAC method was soon used and extended by several researchers, e.g. R. K.-C. Chan and Street (1970a) and R. K.-C. Chan and Street (1970b) who used a surface-heigh function to separate fluid and void, and were able to accurately track a traveling solitary wave. A version of the MAC framework was used by Nichols and Hirt (1975) to simulate free-surface flow around bodies, now locating the interface between fluid and void by using volume tracking (VOF). The volume tracking methodology also enabled using a light fluid and a dense fluid together, to avoid applying free surface boundary conditions.

Purely Lagrangian methods for free-surface flows where also developed, though such methods have obvious problems with large mesh deformations as the vertices follow the flow (Brennen and Whitney 1970; Hirt, Cook, and Butler 1970). Lagrangian methods enabled using the finite element method for free surface flow calculations (Nickell, Tanner, and Caswell 1974). Soon after the arbitrary Lagrangian–Eulerian (ALE) method by Hirt, Amsden, and Cook (1974) solved many of the problems inherent in purely Lagrangian methods as the mesh deformation could now be controlled independently of the fluid-particle velocities.

The ALE method can be used for the study of realistic free-surface flows, giving satisfactory results as long as there is no surface overturning or other

topological changes (Braess and Wriggers 2000; Huerta and W. K. Liu 1988; Hughes, W. K. Liu, and Zimmermann 1981; Ramaswamy and Kawahara 1987). In addition to viscous methods, similar finite element methods for inviscid free-surface flows with moving meshes in potential theory applications were also developed (Cai et al. 1998; Eatock Taylor 1996; Ma and Yan 2006; Robertson and Sherwin 1999), though these do not have a time dependency in the bulk, only on the free surface, due to solving the Laplace equation.

1.4.2 DG FEM for incompressible flow

A discontinuous Galerkin methods for incompressible flow was first presented by Baker, Jureidini, and O. A. Karakashian (1990), who used piecewise solenoidal approximating functions to solve the Stokes equation. They used an interior penalty (IP) method to enforce continuity on the internal facets. This was extended to the Navier–Stokes equations by O. Karakashian and Katsaounis (2000). An interior penalty method was also used by Hansbo and Larson (2002) for the Stokes problem and in Girault, Rivière, and M. Wheeler (2005) for the Stokes and Navier–Stokes equations. J.-G. Liu and Shu (2000) solved the Navier-Stokes equations with discontinuous elements for the vorticity, but a continuous stream-function.

Cockburn et al. developed the local discontinuous Galerkin (LDG) method for the Stokes equation in Cockburn, Kanschat, Schötzau, and Schwab (2002), extended this to the Oseen equations in Cockburn, Kanschat, and Schötzau (2004), and finally to the Navier–Stokes equations in Cockburn, Kanschat, and Schötzau (2005). This thesis is based on the method from 2005, but uses the symmetric interior penalty (SIP) method for the elliptic term rather than LDG, similar to what is described in Cockburn, Kanschat, and Schötzau (2007) and Shahbazi, Fischer, and Ethier (2007). See also Schötzau, Schwab, and Toselli (2003a) and Schötzau, Schwab, and Toselli (2003b) for early uses of IP methods for the Stokes problem.

1.4.3 Slope limiters in DG FEM

Slope limiters for DG FEM were introduced for explicit methods by Chavent and Cockburn (1989). They improved the stability of an existing DG method by Chavent and Salzano (1982), and were able to prove that the slope-limited method was total variation bounded under a CFL restriction, greatly expanding the usefulness of DG methods for convection-dominated problems. This idea was later expanded into the very successful Runge–Kutta (RKDG) based methods for explicit problems, starting with the presentation of the first slope-limited DG method that preserved the accuracy near maxima in Cockburn and Shu (1991).

1.4.4 Recent developments

While the above sections show the beginnings of the fields on which this thesis builds, a lot has happened in recent times. Below follows a summary of some of

the relevant works in this domain from the last (roughly) 15 years. The works have been attempted categorised, to break this review into parts. The works that belong in multiple categories have been put into the one deemed most relevant in the context of this thesis. It should also be noted that the focus of the selected works is very much on *incompressible* flow. A lot has been done in parallel on *compressible* flow, but for brevity and due to the focus on exact incompressibility in this thesis, those works are as a rule not included.

HDG Recent work on DG FEM methods for incompressible flow include the hybridizable DG methods (HDG), where local unknowns may be recovered as a post-processing operator after solving a system with fewer degrees of freedom defined only on the mesh skeleton. The reduced system is created by a lifting operation, similar to static condensation. Cockburn and Gopalakrishnan (2005a) and Cockburn and Gopalakrishnan (2005b) hybridized a velocity-vorticity formulation of the incompressible Stokes equation and obtained an exactly incompressible velocity field. The following year a hybrid LDG method for the velocity-pressure formulation was published by Carrero, Cockburn, and Schötzau (2006). Sharp estimates for this was presented later in Cockburn, Gopalakrishnan, et al. (2010). In 2011, Nguyen, Peraire, and Cockburn extended their work from the previous year on compressible flow to hybridizing the incompressible Navier–Stokes equations. A-priori error estimates for this method were provided by Cesmelioglu, Cockburn, and Qiu (2017). In a *hp*-adaptivity context, Egger and Waluga (2013) provided an analysis of HDG methods for incompressible Stokes flow and used both a-priori and a-posteriori error estimates to adaptively refine meshes in two numerical convergence studies. Qiu and Shi (2016) presented a HDG method of the incompressible Navier–Stokes equations on general polyhedral meshes, and obtained super-convergence for the velocity without the post-processing step necessary in earlier methods. Giorgiani, Fernández-Méndez, and Huerta (2014) developed a *p*-adaptive HDG method for the incompressible Navier–Stokes equations where a local inexpensive a-posteriori error estimator is used to obtain uniform error distribution in the domain by increasing the polynomial degree in under-resolved cells. Rhebergen and G. N. Wells (2018) presented a HDG method for the incompressible Navier–Stokes equations that is pointwise divergence free and $H(\mathrm{div})$ conforming with both velocities and pressures defined on the facets in the hybrid function space. This new method is mass and momentum conserving, energy stable and pressure robust.

Space-time DG The space-time discontinuous Galerkin methods, where DG discretisation is used also for the time-dimension, offers the possibility to construct methods that are conservative in time as well as in space. The methods also make conservation possible in mesh-refinement applications where the treatment of non-matching interfaces can be used in the time-domain. J. J. W. van der Vegt and van der Ven (2002) presented a space-time DG method for the compressible Euler equations. This method is extended in Klaij, J. J. W. van der Vegt, and van der Ven (2006) to viscous flow. Pesch and J. J. W. van der Vegt (2008) also studied

the Euler equations, but extends the method to incompressible flow. Rhebergen, Cockburn, and J. J. van der Vegt (2013) presents a space-time discontinuous method for viscous incompressible flow. A method for solving the incompressible Navier–Stokes equations with a space-time discontinuous method is also provided by Tavelli and Dumbser (2015) and Tavelli and Dumbser (2016), this time on staggered unstructured meshes using isoparametric elements. Staggered meshes for space-time calculations are also used by Fambri and Dumbser (2017), this time with a focus on adaptive mesh refinement. In Fambri (2019) this method is combined with a sub-cell slope limiter to resolve shocks.

Solution methods Solving the Navier–Stokes equations in an efficient manner often involves projection or splitting methods to enable fast time-stepping of segregated velocity and pressure solvers. Girault, Rivière, and M. F. Wheeler (2005) presents a DG version of the fractional-step method by Blasco, Codina, and Huerta (1998). Persson and Peraire (2006) studied low-memory solution of the LDG discretisation of the compressible Navier–Stokes equations using products of structured matrices to represent the unstructured discretised operator matrices, and used the Newton-method to handle the non-linearity. Hesthaven and Warburton (2008) and later Ferrer and Willden (2011) show how the classic pressure-correction method by Karniadakis, Israeli, and Orszag (1991) can be implemented in a DG setting. Botti and Di Pietro (2011) presents a pressure-correction scheme for discontinuous velocities and continuous pressures that is LBB stable for equal-order discretisations. Kopecz (2012) show how a splitting method presented in Turek (1999) can be used with a DG FEM discretisation. In Klein, Kummer, and Oberlack (2013) the classic SIMPLE algorithm is extended to DG FEM and the applicability of the DG SIMPLE method is shown in several numerical experiments. Steinmoeller, Stastna, and Lamb (2013) implement a standard differential pressure-correction scheme for the DG discretisation of the incompressible Navier–Stokes equations, but supplement the solution procedure with a local cell-wise projection into an exactly solenoidal vector function space, and hence obtain exact incompressibility. Lehrenfeld and Schöberl (2016) contribute an operator-split solution method for the HDG discretisation of the incompressible Navier–Stokes equations. Pandare and Luo (2016) do the same with a pressure-correction method for a (reconstructed) discontinuous velocity and a continuous pressure, and satisfy the LBB condition with an equal-order HDG discretisation. An incremental pressure-correction scheme for the DG solutions of the incompressible Navier–Stokes equations that is stable for very small time-steps, where previous schemes have failed, is presented by Emamy et al. (2017). They show numerical results indicating that the scheme is long-term stable and accurate also for equal-order elements. Fehn, W. A. Wall, and Kronbichler (2017) also study instabilities in pressure-correction schemes related to small time steps, and find that even when including LBB/inf–sup stabilising terms, there are still instabilities for equal-order discretisations. Piatkowski, Müthing, and Bastian (2018) use a rotational incremental pressure-correction scheme with a Raviart–Thomas post-processing procedure that gives

excellent mass-conservation.

CPU-efficiency As DG methods are reaching the mainstream and are applicable for many practical applications there are several researchers who work on improving the computational performance and scalability. One very successful avenue is the work on sum-factorised matrix-free DG methods, see e.g Krank et al. (2017) for recent developments in their method where the wall-clock time per time step is almost independent on the polynomial degree, thanks to an entirely matrix-free implementation. Fehn, W. A. Wall, and Kronbichler (2018a) focus on efficiency on under-resolved problems and demonstrate that the wall-clock time can be reduced by an order of magnitude by smart use of matrix-free methods. In Fehn, W. A. Wall, and Kronbichler (2018b) they focus on using a local Lax–Friedrichs flux to stabilise under-resolved turbulent simulations. Kronbichler and W. Wall (2018) compares continuous and discontinuous methods, and find that matrix-free sum-factorised DG methods can be almost as fast as continuous methods, and significantly faster than the hybridized version of the DG scheme (and the statically condensed continuous scheme), even though the number of unknowns is much larger. Kronbichler and Kormann (2019) compares the method to the arithmetic performance of modern CPUS, and find that it reaches 60% of the theoretical peak. Consequently, parts of the code other than the matrix solver are starting to dominate, in stark contrast to matrix-based methods which typically spend the overwhelming majority of their time inside the linear-algebra solver. Fehn, Munch, et al. (2019) show that the matrix-free methods can be combined with specially crafted multi-grid preconditioners that, as part of the multi-grid cycle, project from DG to CG and hence become independent of the magnitude of the penalty term, which makes the original problem worse conditioned when the penalty is increased. Another way to increase performance is by vectorisation and use of GPU computing, see e.g. Einkemmer and Wiesenberger (2014).

Solenoidal elements Several researchers have been working on new elements and other techniques targeting exactly solenoidal velocity fields. O. Karakashian and Katsaounis (2006) use an analytically solenoidal vector function space to represent the velocity. The same is done by Montlaur et al. (2010), but now the system is solved first for hybrid pressures (on the mesh skeleton), and then locally for the full pressure afterwards. A more traditional approach is taken by Mozolevski, Süli, and Bösing (2006), who solve the incompressible Navier–Stokes equations in the fourth-order stream-function formulation, also guaranteeing exact incompressibility. The stream function approach is also studied by Hansbo and Larson (2008), who test three different approaches to eliminating the pressure from the equations. Labeur and G. N. Wells (2012) use a Lagrange-multiplier to enforce continuity in the fluxes in a generalised hybrid DG method, and are able to conserve both mass and momentum with fewer degrees of freedom than standard DG methods. (Cheung et al. 2015) show a new staggered mesh method.

They obtain local and global conservation, and a strongly divergence free velocity field that super-converges after post-processing.

VOF Work on finite element methods used in conjunction with the volume-of-fluid method for free-surface flows include Jeong and Yang (1998), who use VOF to track the free surface and also to determine where to impose boundary conditions, i.e. not solving the flow equations in the air-phase. Cervone, Manservisi, and Scardovelli (2010) uses FEM and VOF to study a liquid spray into a gas environment at low Reynolds numbers. Sun and Kang (2012) shows a numerical wave tank, also with FEM and VOF, and uses this to study wave-breaking on a sloping beach. There are also several recent publications on combining FEM and level-set methods free-surface flow simulation, see e.g. Tornberg and Engquist (2000), Groß, Reichelt, and Reusken (2006), Groß and Reusken (2007), and Marchandise and Remacle (2006). There are also authors working on improving the mass-conserving properties of level-set methods by use of DG FEM without coupling to VOF, see e.g. Di Pietro, Lo Forte, and Parolini (2006), Grooss and Hesthaven (2006), Karakus et al. (2016), Owkes and Desjardins (2013), and Pochet et al. (2013).

Other Other recent advances include, e.g., penalty-free elliptic terms (Riviere and Sardar 2014), grad–div-like stabilization (Akbas et al. 2017) and pressure-robust solvers (Schroeder and Lube 2017; Schroeder and Lube 2018). Meshes with non-matching interfaces (hanging nodes) at subdomain boundaries can be very useful for (re-)meshing, increasing the resolution in sub-domains to capture local features without re-meshing the surrounding elements to obtain matching nodes (Girault, Rivière, and M. Wheeler 2005; Rivière and Girault 2006). Related to re-meshing, Baiges et al. (2017) uses ALE as a sub-step, after which the solution is projected back to a fixed mesh at the end of each time step, called fixed-mesh ALE.

1.5 The discontinuous Galerkin method

This section gives an introduction to the discontinuous Galerkin finite element method (DG FEM). This presentation will be more thorough than what is found in the following papers, and is more suitable for beginners in the DG FEM method. For ease of reference the nomenclature is summarised in table 1.1. Note that there may be some minor differences between the symbols used here and what is used later in the attached papers.

Table 1.1: Nomenclature.

Symbol	Definition
D	The number of spatial dimensions
N	The polynomial degree of the basis functions
H	The total number of elements
Ω	The domain, a closed space in $\mathbb{R}^D$
$\partial\Omega$	The boundary of the domain
$\mathcal{T}$	The tessellation/triangulation of Ω into H cells
h	A typical cell size
$\square_h$	A discretised version of $\square$, i.e. $\Omega_h = \mathcal{T}$
i	Global index, $i \in [1, H]$
j	Local index, $j \in [1, N]$
K_i	A mesh cell
∂K_i	The boundary of K_i
F	A facet of a cell. $\partial K = \bigcup F_j$
n_j	A mesh cell node / vertex
d_j	A degree of freedom
ϕ_j	A basis function
$P_N(K)$	The space of polynomials of order N on K
Γ	All cell facets, $\Gamma = \bigcup \partial K$
Γ_D	All external facets, the tessellation of $\partial\Omega$
Γ_0	All internal facets, $\Gamma_0 = \Gamma \setminus \Gamma_D$

Let us start with a simple approximation problem. We want to approximate a function f on a domain $\Omega \in \mathbb{R}^D$ enclosed by the surface $\partial\Omega$. The approximation f_h will be discrete, i.e. it will be possible to describe the approximation of f by a finite number of scalar numbers, called degrees of freedom.

A triangulation—or more generally a tessellation—of the domain Ω is first created by a suitable algorithm. A good choice is to use a software such as Gmsh (Geuzaine and Remacle 2009). The tessellation, $\mathcal{T}$, is made up of non-overlapping cells K with boundaries ∂K. There are H cells in $\mathcal{T}$ and the average

characteristic size of the cells is h. We require

$$\lim_{h\to 0} \mathcal{T} = \lim_{H\to\infty} \bigcup_{i=0}^{H} K_i = \Omega. \tag{1}$$

The discrete approximation f_h to the continuous function f is said to be *consistent* if it approaches f as H approaches infinity,

$$\lim_{H\to\infty} f_h = \lim_{h\to 0} f_h = f. \tag{2}$$

Internally in each cell K we approximate the function f by polynomial basis functions of order N in D dimensions, $f_h \in P_N(K)$. Each individual cell has its own complete set of basis functions and a corresponding set of degrees of freedom. The degrees of freedom are the weights of the basis functions. In the discontinuous Galerkin method the basis functions only span one cell, so the degrees of freedom are not shared among neighbouring cells. This means that there may be discontinuities in the approximated functions across the inter-cell faces.

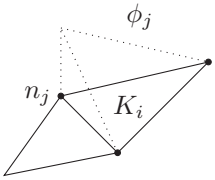


Figure 1.2: A linear nodal basis function.

In the following text a *finite element* should be taken to mean the combination of a cell, here triangles or tetrahedra, with basis functions, here discontinuous polynomial spaces, and a way to scale the basis functions to approximate an arbitrary function, the degrees of freedom. This is the classical definition of a finite element by Ciarlet (1976). We define the basis functions $\phi_j(\boldsymbol{x})$ such that they are unity at node n_j and zero at all other nodes, see figure 1.2. This is called a nodal basis. For each node n_j in K there will be one basis function ϕ_j and one degree of freedom d_j. To form a proper finite element we require unisolvency: if all degrees of freedom are zero then the combined scaled basis functions must be identically zero everywhere, they are fully constrained by the degrees of freedom.

To evaluate the value of the approximated function f_h at an arbitrary location $\boldsymbol{x}$ in $\mathcal{T}$, we must first find which element K_i contains $\boldsymbol{x}$ and then calculate

$$f(\boldsymbol{x}) \approx f_h(\boldsymbol{x}) = \sum_{j=1}^{N} \phi_j(\boldsymbol{x})\, d_j. \tag{3}$$

With a nodal basis, the value of each degree of freedom d_j will be the value of the approximated function f_h at node n_j. The value of the true function f

at the node may be different, it is possible to select the weights d_j to minimise e.g. the L_2 norm, $\|f - f_h\|_{L_2}^2 = \int_\Omega (f - f_h)^2 \, \mathrm{d}\boldsymbol{x}$, instead of selecting to perfectly interpolate f at the nodes. This will lead to discontinuities in f_h between elements unless the function f can be perfectly approximated by the selected finite elements—if the function f is a polynomial of order N or lower, then the minimum error norm $\|f - f_h\|_{L_2}$ will be zero and $f = f_h$ everywhere.

Let us now look at two elements in $\mathcal{T}$, K^+ and K^-. The two elements share a common face F, see figure 1.3. Which element is denoted K^+ and which is denoted K^- is arbitrary and not related to positions of the two elements relative to the coordinate system origin or any other systematic ordering of elements.

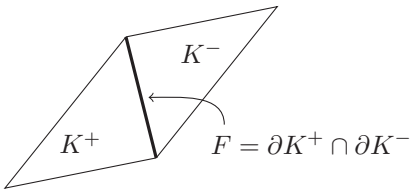


Figure 1.3: Two elements sharing a face.

We are interested in working with the discontinuities, or jumps, across the shared face F in our formulation of the discontinuous Galerkin finite element method. We define the jump $[\![a]\!]$ in a quantity a between a^+ on the K^+ side and a^- on the K^- side as,

$$[\![a]\!] = a^+ - a^-, \tag{4}$$

$$[\![a]\!]_{\boldsymbol{n}} = a^+ \cdot \boldsymbol{n}^+ + a^- \cdot \boldsymbol{n}^-, \tag{5}$$

where the outward normal vector from K^+ on F is denoted $\boldsymbol{n}^+$ and equation (5) contains dot products only if a is a vector, otherwise it is a product of a scalar and a vector. The second operator needed to work with the discontinuities is the average. The average value $\{\{a\}\}$ of a on the shared face F is simply defined as

$$\{\{a\}\} = \frac{1}{2}(a^+ + a^-). \tag{6}$$

These definitions lead to the following identity which will be used later,

$$[\![ab]\!] = [\![a]\!] \{\{b\}\} + [\![b]\!] \{\{a\}\}. \tag{7}$$

Integrals over the domain Ω are approximated as a sum of integrals over each element, $K \in \mathcal{T}$. This allows using integration by parts to introduce integrals over the element boundaries. These integrals will be used to couple the unknown in each element with the unknowns in the neighbouring elements. As a reminder, integration by parts can be written in single and multi-dimensional form as

$$\int_{x_0}^{x_1} ab' \, \mathrm{d}x = [ab]_{x_0}^{x_1} - \int_{x_0}^{x_1} a'b \, \mathrm{d}x, \tag{8}$$

$$\int_\Omega \boldsymbol{a} \cdot \nabla b \, \mathrm{d}\boldsymbol{x} = \int_{\partial\Omega} \boldsymbol{a}b \cdot \boldsymbol{n} \, \mathrm{d}s - \int_\Omega \nabla \cdot \boldsymbol{a}b \, \mathrm{d}\boldsymbol{x}. \tag{9}$$

After integration by parts has introduced facet integrals over ∂K in the global sum,

$$\int_\Omega \cdot \,\mathrm{d}\boldsymbol{x} = \sum_{K_i \in \mathcal{T}} \int_{K_i} \cdot \,\mathrm{d}\boldsymbol{x} = \sum_{K_i \in \mathcal{T}} \left[\int_{\partial K_i} \cdot \,\mathrm{d}s - \int_{K_i} \cdot \,\mathrm{d}\boldsymbol{x} \right], \tag{10}$$

the sum will end up containing integrals over each internal facet twice, once for each of the elements sharing the facet. This global sum of integrals will be transformed into a sum over all elements, $K \in \mathcal{T}$, a sum over all external faces, $F \in \partial\Omega$, and a sum over all internal faces, $F = \partial K^+ \cap \partial K^-$. For computational efficiency it is beneficial to include each entity in this sum only once—not sum over all cells and then over each cell's connected facets—so we will need to describe the total contribution from the duplicated internal element boundary integrals. This will be described in the next section by the introduction of *numerical fluxes*, a key part of the discontinuous Galerkin method.

1.5.1 The advection equation

The advection equation is also known as the transport equation or the convection equation. The equation for advection of a conserved scalar field c in a known velocity field $\boldsymbol{u}$ with no source terms for c is

$$\frac{\partial c}{\partial t} + \nabla \cdot (\boldsymbol{u} c) = 0. \tag{11}$$

If we restrict ourselves to an advecting velocity field that is divergence free, $\nabla \cdot \boldsymbol{u} = 0$, we can write

$$\frac{\partial c}{\partial t} + \boldsymbol{u} \cdot \nabla c = 0. \tag{12}$$

Let us convert the advection equation to its weak form by approximating the unknown function by a DG finite element, $c \in P_N(K)$, multiplying with a DG test function, $v \in P_N(K)$, and integrating over the domain. We can split the integral into sub-integrals over each element. For each element we then have the following integral, where $\mathrm{d}\boldsymbol{x}$ means a volume integral in all the element's D spatial dimensions,

$$\int_K \frac{\partial c}{\partial t} v \,\mathrm{d}\boldsymbol{x} + \int_K \boldsymbol{u} \cdot \nabla c v \,\mathrm{d}\boldsymbol{x} = 0. \tag{13}$$

One obvious problem with this is that we have no equations linking the element with its neighbours due to the discontinuous formulation. Using equation (9) on the second integral in equation (13) with $\boldsymbol{a} = \boldsymbol{u}v$, $b = c$ and using $\nabla \cdot \boldsymbol{u} = 0$ we get

$$\int_K \frac{\partial c}{\partial t} v \,\mathrm{d}\boldsymbol{x} + \int_{\partial K} (c\boldsymbol{u}) \cdot \boldsymbol{n} \, v \,\mathrm{d}s - \int_K c\boldsymbol{u} \cdot \nabla v \,\mathrm{d}\boldsymbol{x} = 0 \tag{14}$$

Now the integral over the element boundary ∂K in equation (14) can be used to establish a link between the degrees of freedom in each element. If we look at

the integral over a shared facet, $F = \partial K^+ \cap \partial K^-$, the sum of the contributions from K^+ and K^- can be written as a jump since $\boldsymbol{n}^+ = -\boldsymbol{n}^-$,

$$\int_F (c^+ \boldsymbol{u}^+) \cdot \boldsymbol{n}^+ \, v^+ \, \mathrm{d}s + \int_F (c^- \boldsymbol{u}^-) \cdot \boldsymbol{n}^- \, v^- \, \mathrm{d}s = \int_F [\![\, cv\boldsymbol{u} \,]\!] \cdot \boldsymbol{n}^+ \, \mathrm{d}s. \tag{15}$$

Let us use our physical intuition and require that the flux $\widehat{\boldsymbol{u}c}$ is the same on both sides of the shared facet F. This means that $\widehat{\boldsymbol{u}c}^+ = \widehat{\boldsymbol{u}c}^- = \widehat{\boldsymbol{u}c}$, and we can write the integral with only a jump in the test function v,

$$\int_F [\![\, cv\boldsymbol{u} \,]\!] \cdot \boldsymbol{n}^+ \, \mathrm{d}s = \int_F \widehat{\boldsymbol{u}c} \cdot \boldsymbol{n}^+ \, [\![\, v \,]\!] \, \mathrm{d}s = \int_F (\widehat{\boldsymbol{u}c} \cdot \boldsymbol{n})^+ \, [\![\, v \,]\!] \, \mathrm{d}s. \tag{16}$$

What remains is to decide on an appropriate flux $(\widehat{\boldsymbol{u}c} \cdot \boldsymbol{n})^+$. A strict requirement is that in the case where $\boldsymbol{u}$ and c are continuous across F, then the flux must reduce to this continuous value. Otherwise, we are free to select the flux that gives us the best combination of accuracy and stability properties. The continuity of the flux ensures consistency of the numerical scheme, and having a single valued flux on the shared face F ensures conservation of c, see for example Cockburn (2003).

To start defining the flux, it will help to first define an upwind normal velocity $u_{\boldsymbol{n}\mathrm{U}}$, and a downwind normal velocity $u_{\boldsymbol{n}\mathrm{D}}$,

$$u_{\boldsymbol{n}\mathrm{U}} = \frac{1}{2}(\boldsymbol{u} \cdot \boldsymbol{n} + |\boldsymbol{u} \cdot \boldsymbol{n}|), \tag{17}$$

$$u_{\boldsymbol{n}\mathrm{D}} = \frac{1}{2}(\boldsymbol{u} \cdot \boldsymbol{n} - |\boldsymbol{u} \cdot \boldsymbol{n}|). \tag{18}$$

In the case where $\boldsymbol{u}^+$ is pointing outwards on F then $u^+_{\boldsymbol{n}\mathrm{U}} > 0$ and $u^+_{\boldsymbol{n}\mathrm{D}} = 0$ on F. This can be used to define a flux with a blending parameter β where $\beta = 0$ gives a pure upwind flux, $\beta = 0.5$ gives a central flux while $\beta = 1.0$ gives a pure downwind flux,

$$(\widehat{\boldsymbol{u}c} \cdot n)^+ = (1 - \beta) \, [\![\, c \, u_{\boldsymbol{n}\mathrm{U}} \,]\!] + \beta \, [\![\, c \, u_{\boldsymbol{n}\mathrm{D}} \,]\!] \, . \tag{19}$$

Many numerical schemes for advection exist for the case where c is a piecewise constant function with a single value in each element. Such flux-limiting schemes ensure stability (see section 1.6), and can be used for surface capturing (see section 1.8). For higher-order methods it is common to use simple flux limiters, such as a pure upwind flux, together with slope limiters to ensure stability in the presence of shocks/jumps (see section 1.7). Pure upwind fluxes are used in the included papers in conjunction with the slope-limiting strategy from Paper I when using higher-order elements for advection of the velocity in the Navier–Stokes momentum equation.

UFL implementation of advection

An implementation of an advection equation in the Unified Form Language (UFL), used for describing weak forms in FEniCS (Alnæs 2012; Alnæs et al.

2014), is given below, Here $\mathrm{d}s$ denotes integrals over the external boundary $\partial\Omega$ while $\mathrm{d}S$ denotes integrals over internal faces Γ_0,

Listing 1.1: Advection with blended flux in FEniCS UFL.

```
# Upwind and downwind fluxes in direction normal to the face
flux_nU = c * (inner(vel, normal) + abs(inner(vel, normal))) / 2
flux_nD = c * (inner(vel, normal) - abs(inner(vel, normal))) / 2

# Define the blended flux
# The blending factor beta is continuous, so beta('+') == beta('-')
b = beta('+')
flux = (1 - b) * (flux_nU('+') - flux_nU('-')) \
           + b * (flux_nD('+') - flux_nD('-'))

# Equation to solve
eq = (c - c_prev) / dt * v * dx \
     - c * inner(vel, grad(v)) * dx \
     + flux * jump(v) * dS \
     + c * inner(vel, normal) * v * ds
```

1.5.2 Elliptic operators

The discretisation of elliptic operators with the discontinuous Galerkin method is not as straight forward as for hyperbolic operators such as convection. There is no direction to diffusion, so up-winding is not possible. Instabilities arise related to checker-boarding with very different solutions in neighbouring cells, so there is a requirement to somehow penalise large jumps in the solution.

The symmetric interior-penalty (SIP) method by Arnold (1982) is what we will use below to arrive at a stable weak form. This is an extension of Nitsche's (1971) method for constructing stable weak boundary conditions, i.e. boundary conditions that are a part of the weak form and not strongly applied to the matrix after assembly.

Let us start by looking at an elliptic term, $\nabla^2 c$, which is multiplied by a test function, v, before the result is integrated over the domain,

$$\int_\Omega \nabla \cdot (\nabla c) \cdot v \, \mathrm{d}\boldsymbol{x}. \tag{20}$$

After splitting the integral into integrals over each element and then performing integration by parts, the result, for one element, is

$$\int_{\partial K} v \nabla c \cdot \boldsymbol{n} \, \mathrm{d}s - \int_K \nabla c \cdot \nabla v \, \mathrm{d}\boldsymbol{x}. \tag{21}$$

When we sum equation (21) over all elements, the sum of integrals over an internal facet with contributions from both ∂K^+ and ∂K^- is written as a jump, similar how it was done in equation (15),

$$\int_F [\![v \nabla c]\!] \cdot \boldsymbol{n}^+ \, \mathrm{d}s. \tag{22}$$

The identity in equation (7) is then applied to equation (22) giving

$$\int_F [\![v]\!] \{\{\nabla c\}\} \cdot \boldsymbol{n}^+ \,\mathrm{d}s + \int_F [\![\nabla c]\!] \{\{v\}\} \cdot \boldsymbol{n}^+ \,\mathrm{d}s \tag{23}$$

The SIP method is used to minimise the inter-element jumps and the deviation from the Dirichlet boundary conditions, $c = g$ on Γ_D. The constraints

$$[\![c]\!] = 0 \text{ on } \Gamma_0 \tag{24}$$

$$(c - g) = 0 \text{ on } \Gamma_D \tag{25}$$

are enforced weakly by the introduction of the symmetric interior-penalty test function, $\tilde{v} = \kappa [\![v]\!] - \{\{\nabla v\}\} \cdot \boldsymbol{n}^+$, where κ is a penalty parameter which must be sufficiently large to ensure stability. After integrating equations (24) and (25) over the domain, the resulting two integrals are added to the weak form,

$$\int_{\Gamma_0} [\![c]\!] \tilde{v} \,\mathrm{d}s + \int_{\Gamma_D} (c - g)\tilde{v} \,\mathrm{d}s = 0. \tag{26}$$

The final stabilised weak form of the elliptic operator is symmetric due to the choice of penalty test function. The weak form has one cell integral and seven facet integrals,

$$\begin{aligned} &- \int_{\mathcal{T}} \nabla c \cdot \nabla v \,\mathrm{d}\boldsymbol{x} + \int_{\Gamma_D} v \nabla c \cdot \boldsymbol{n} \,\mathrm{d}s \\ &+ \int_{\Gamma_0} [\![v]\!] \{\{\nabla c\}\} \cdot \boldsymbol{n}^+ \,\mathrm{d}s + \int_{\Gamma_0} [\![\nabla c]\!] \{\{v\}\} \cdot \boldsymbol{n}^+ \,\mathrm{d}s \\ &+ \int_{\Gamma_0} [\![c]\!] \{\{\nabla v\}\} \cdot \boldsymbol{n}^+ \,\mathrm{d}s - \int_{\Gamma_0} \kappa [\![c]\!] [\![v]\!] \,\mathrm{d}s \\ &+ \int_{\Gamma_D} (c - g) \nabla v \cdot \boldsymbol{n} \,\mathrm{d}s - \int_{\Gamma_D} \kappa (c - g) v \,\mathrm{d}s, \end{aligned} \tag{27}$$

where the first two lines are from integration by parts of the elliptic operator and the next two lines stabilise the internal and external facets respectively. On the external boundary, $[\![v]\!] = \{\{v\}\} = v$ has been used to get the last line.

It is common to use twice as much penalisation, 2κ, at the boundary facets, $F \in \Gamma_D$, compared to the internal facets, $F \in \Gamma_0$. See Epshteyn and Rivière (2007) and Shahbazi, Fischer, and Ethier (2007) for more details on selection of the penalty parameters and Arnold et al. (2002) for an overview of other methods to stabilise elliptic operators in DG methods. All papers in this thesis have used the symmetric interior penalty method.

UFL implementation of diffusion

The UFL code in listing 1.2 describes the Poisson problem

$$\begin{aligned} -\nabla^2 c &= f \qquad \text{in } \Omega, \\ c &= g \qquad \text{on } \partial\Omega. \end{aligned} \tag{28}$$

The a and L variables contain the bilinear and linear forms respectively. The $\mathrm{d}S$ measure integrates over Γ_0 while $\mathrm{d}s$ integrates over Γ_D. The trial function, c, and the test function, v, are both scalar and can be represented by DG FEM elements, though the code will also work for continuous elements where $[\![c]\!] = [\![v]\!] = 0$ by definition.

Listing 1.2: DG FEM diffusion with SIP in FEniCS

```
# The interior of the domain
a = dot(grad(u), grad(v)) * dx
L = f * v * dx
a -= dot(n('+'), avg(grad(c))) * jump(v) * dS
a -= dot(n('+'), avg(grad(v))) * jump(c) * dS
a += kappa * jump(c) * jump(v) * dS

# Dirichlet boundary conditions
a -= dot(n, grad(c)) * v * ds
a -= dot(n, grad(v)) * c * ds
L -= dot(n, grad(v)) * g * ds
a += 2 * kappa * c * v * ds
L += 2 * kappa * g * v * ds
```

1.6 Convective stability in piecewise constant methods—flux limiters

Convective stability in low-order methods can be ensured by the use of a flux limiter. This limiter stabilises the average value in each cell, which is sufficient for convective stability in numerical methods that approximate the convected value by piecewise constant functions. Low-order finite volume methods and the lowest-order discontinuous Galerkin method are common examples of such methods. To follow the flux-limiter literature, we will here study the transport of a scalar function ϕ from a central cell K_C to a downwind cell K_D by the convecting velocity $\boldsymbol{u}$. The cells are separated by the facet F. Upwind of the central cell there is an upwind cell K_U. A sketch is shown in figure 1.4.

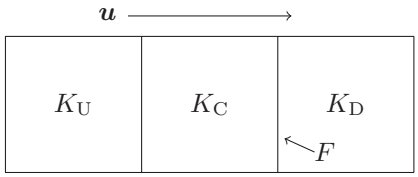


Figure 1.4: Upwind, central and downwind cells.

A situation where the scalar function ϕ is increasing from K_C to K_D is shown in figure 1.5. The convective-boundedness criterion (CBC) by Leonard (1979) can be used to ensure convective stability. The CBC has two requirements for convective stability: (i) the value of ϕ must change monotonically between cells, and (ii) the flux must be pure upwind when ϕ_C is a local extremum in relation

to ϕ_{U} and ϕ_{D}. The first criterion is directly linked to avoiding boundedness problems, as a reconstructed value between two known cell-average values must be bounded by the neighbouring cell averages. If the flux over- or under-shoots the neighbouring values this could create a new global minimum or maximum, violating global boundedness and potentially leading to negative densities or other obviously unphysical results. The second criterion selects the most diffusive flux near existing local maxima. Using a diffusive flux will in make sharp peaks smoother and lower, combatting unphysical spikes. Recent work by e.g. Ping-Li, Wen-Quan, and Mao-Zheng (2003) and Yu et al. (2001) confirms that the first criterion in sufficient and necessary, but has questioned whether the second criterion is only sufficient, but may not in fact be necessary, and as such less diffusive fluxes can be chosen. However, Chourushi (2019) indicates that such extensions in the non-monotonic range "gives rise to loss of numerical stability for convection dominated fluid flows".

Here we present the original CBC as explained in the two criteria above. See also Gaskell and Lau (1988) for a detailed discussion of the CBC. Only the first requirement is relevant in the case shown in figure 1.5, which implies that any choice of flux satisfying $\phi_{\mathrm{C}} \leq \phi_F \leq \phi_{\mathrm{D}}$ will be stable. If we had $\phi_{\mathrm{U}} > \phi_{\mathrm{C}}$ in figure 1.5 then ϕ_{C} would be a local minimum, and we would have to require $\phi_F = \phi_{\mathrm{C}}$.

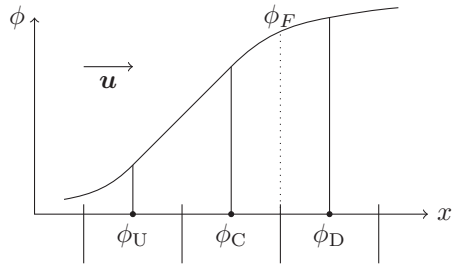


Figure 1.5: The function ϕ at cell centers and at the facet F.

To construct a stable flux limiter according to the CBC, let us first introduce the normalised quantities $\tilde{\phi_{\mathrm{C}}}$ and $\tilde{\phi_F}$,

$$\tilde{\phi_{\mathrm{C}}} = \frac{\phi_{\mathrm{C}} - \phi_{\mathrm{U}}}{\phi_{\mathrm{D}} - \phi_{\mathrm{U}}}, \tag{29}$$

$$\tilde{\phi_F} = \frac{\phi_F - \phi_{\mathrm{U}}}{\phi_{\mathrm{D}} - \phi_{\mathrm{U}}}. \tag{30}$$

The flux ϕ_F can be computed by combining the normalised expressions in equations (29) and (30) and eliminating ϕ_{U},

$$\phi_F = (1 - \tilde{\beta})\phi_{\mathrm{C}} + \tilde{\beta}\phi_{\mathrm{D}}, \tag{31}$$

where $\tilde{\beta}$ can be interpreted as a downwinding factor,

$$\tilde{\beta} = \frac{\tilde{\phi_F} - \tilde{\phi_C}}{1 - \tilde{\phi_C}}. \tag{32}$$

Setting $\tilde{\phi_F} = 1$ gives $\tilde{\beta} = 1$ and a pure downwind flux. Taking $\tilde{\phi_F} = \tilde{\phi_C}$ results in $\tilde{\beta} = 0$ and a pure upwind flux. The normalised-variable diagram (NVD) by Leonard (1988) in figure 1.6 shows a shaded region where the facet value ϕ_F satisfies the CBC. For $\tilde{\phi_C} \notin [0, 1]$ only the upwind scheme is stable, while for $\tilde{\phi_C} \in [1, 0]$ it is possible to blend the upwind scheme and the downwind scheme.

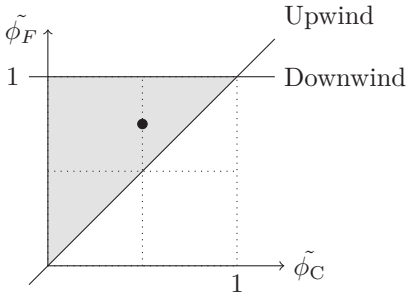


Figure 1.6: Normalised Variable Diagram.

Higher-order DG methods can give higher-order spatial convergence even with a pure upwind flux, due to the non-constant approximating polynomials inside each cell. Piecewise-constant methods can also give higher-order convergence with an appropriate flux. A convection scheme that passes through the point (0.5, 0.75) in the normalised variable diagram with a finite slope will be at least second-order, it will be third-order if the slope is $\frac{3}{4}$ at this point (Leonard 1988). As can be seen, neither the upwind nor the downwind schemes marked in the figure are second-order.

An alternative diagram to classify convection schemes is the Ψ-r diagram by Sweby (1984) shown in figure 1.7. The Sweby diagram is often used to describe the total-variation-diminishing (TVD) schemes that have the property that they do not increase the total variation from one time step to another. For a discretised scalar function ϕ, the total variation is defined as

$$\mathrm{TV}(\phi) = \sum_{i=0}^{N_{\mathrm{elem}}} |\phi_i - \phi_{i-1}|, \tag{33}$$

where e.g. $\phi_i = \phi_C$ and $\phi_{i+1} = \phi_D$.

Sweby defines the face value ϕ_F and the upwind-to-downwind ratio r as

$$\phi_F = \phi_C + \frac{1}{2}\Psi(r)(\phi_D - \phi_C), \tag{34}$$

$$r = \frac{\phi_C - \phi_U}{\phi_D - \phi_C}. \tag{35}$$

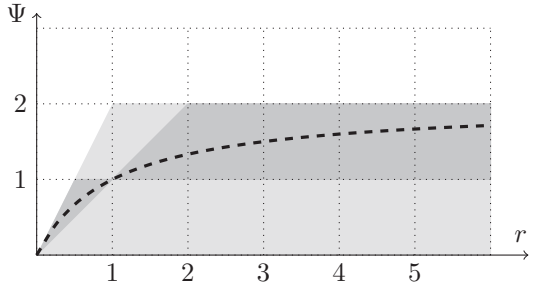


Figure 1.7: The Sweby (1984) diagram. The light-shaded region is the TVD region while the dark-shaded is the second-order TVD region. The dashed line is a second-order TVD scheme by van Leer (1974), see equation (38).

It can easily be seen that $\Psi(r) = 0$ corresponds to an upwind scheme, while $\Psi(r) = 2$ corresponds to a downwind scheme. Leonard (1991) gives the relations between NVD and the TVD diagram,

$$\Psi = \frac{\tilde{\phi_F} - \tilde{\phi_C}}{\frac{1}{2}(1 - \tilde{\phi_C})}, \tag{36}$$

$$r = \frac{\tilde{\phi_C}}{1 - \tilde{\phi_C}}. \tag{37}$$

Leonard (1991) also shows that the TVD region of the normalised-variable diagram is the part of the shaded region in figure 1.6 that falls below $\tilde{\phi_F} = 2\tilde{\phi_C}$. This region is shown in figure 1.8. Constructing a stable flux limiter is now a matter of describing a continuous single-valued function in the CBC diagram that goes from the origin and up to $(1, 1)$. If the function stays inside the TVD region it will satisfy both the CBC and the TVD criteria. A second-order TVD convection scheme should smoothly blend between upwind and downwind schemes in such a way that it stays within the TVD region and passes through the "second-order" point (0.5, 0.75) with a finite slope. One much used second-order flux limiter is the scheme by van Leer (1974). The van Leer flux limiter is shown in figures 1.6 and 1.8, and the expression for the line is

$$\Psi(r) = \frac{r + |r|}{1 + r}. \tag{38}$$

As can be seen from the figures, the TVD region falls inside the CBC region and the CBC criterion is more general than the TVD criterion in terms of the local boundedness (Jasak 1996). See, e.g. Jasak, H. Weller, and Gosman (1999) for information about using the TVD and NVD diagrams to create stable convection schemes on general unstructured meshes.

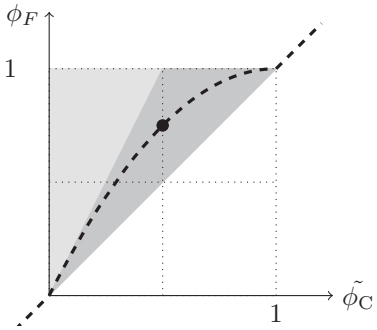


Figure 1.8: Leonard's normalised-variable diagram showing the TVD region and the van Leer (1974) scheme as an example of a second-order TVD scheme.

1.7 Convective stability in higher-order DG methods—slope limiters

The first step to constructing a non-linearly stable higher-order convection scheme is to ensure linear convective stability by selecting an approximate flux. Using a stable flux—such as a pure upwind flux—ensures that the cell averages satisfy the convective-boundedness criterion, CBC.

Higher-order shape functions inside each cell can lead to non-monotonic solutions between two cells, breaking the local maximum principle (Cockburn and Shu 1989). In figure 1.9 the highlighted cell has a cell-average value that is in between the cell averages of the neighbouring cells, but the local polynomial breaks the maximum principle by being outside the bounds created by the maximum and minimum of the local cell averages. The dotted line shows the maximum slope in the cell that satisfies the local maximum principle.

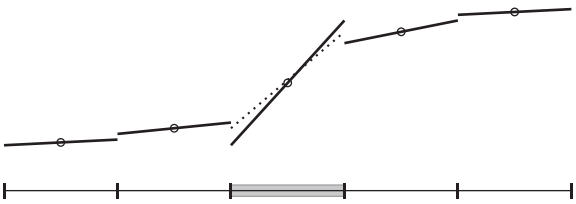


Figure 1.9: A linear function which breaks the local maximum principle.

Methods such as spectral filtering, explicit slope-limiting projections, or adding implicit non-linear diffusion to the weak form can be used to avoid overshoots and force the unknown function to observe the maximum principle. Michoski et al. (2016) provides a comparison of these methods. Explicit slope limiting is the method that has been used in the papers included in this thesis. The implementation details on using slope-limiting methods for solenoidal vector fields such as the fluid velocities is the main topic of Paper I, and the methods

explored there are also used to run realistic 3D two-phase flow simulations in Paper III. These papers show that without slope limiting the simulations blow up almost immediately. With correctly implemented slope limiters there have been no convective stability problems in any of the two-phase benchmark tests.

A slope limiter is a post-processing projection that removes violations of the maximum principle while leaving smooth areas with no such violations alone, thus keeping the overall high-order convergence rate of the numerical method. The limiter is applied to the solution of the discretised version of the weak problem. Solving the weak problem is a linear, possibly linearised, operator, while the slope limiter is a fully non-linear operator that acts directly on the degrees of freedom (DOFs) in each cell. In the 1D case shown in figure 1.9, a slope limiter will typically produce the same function values, except for replacing the DOFs in the highlighted cell by those shown in the dotted line.

The slope-limiting strategy in 2D and 3D used in the included papers is very similar to what is described for 1D above. A linear function can be recast as a mean value plus the derivatives in each of the axis directions. By using this Taylor-polynomial approach, the slopes can be limited such that all the vertex values are bounded by the mean value in the connected cells. For the central vertex shown in figure 1.10, the cell averages of the grey cells will be used to find the maximum and minimum allowable value at the vertex. The approximated function will be multi-valued at the vertex, since all the connected cells have separate DOFs here, but all the individual DOF values will be in the allowable range after slope limiting.

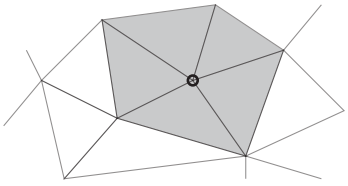


Figure 1.10: Vertex cell neighbours in a 2D triangulated mesh.

For higher-order derivatives, the mean lower-order derivatives in the surrounding cells can be calculated and the same procedure can be used (Kuzmin 2010). This works well inside the domain, especially combined with the criterion that low-order derivatives, or the function values themselves, shall not be limited if the higher-order derivatives are within bounds. This allows smooth extrema where the derivative is within the bounds of the neighbouring cells while the function itself is not.

The main challenge with using slope limiters is related to the handling of boundaries. To demonstrate this, assume that the unknown function has Dirichlet boundary conditions at a wall, and that the first derivative of the function looks like what is shown in figure 1.11. Since the value of the derivative is not known at the boundary—this would require us to specify both Dirichlet and Neumann boundary conditions on the same facet which is not reasonable—the

slope limiter must remove the second derivative near the wall. This is the only way to ensure that the local maximum principle is kept for the first derivative without additional knowledge about what would be a physically reasonable value.

The lack of neighbours at boundaries is a fundamental problem with the method, so some loss of accuracy must be expected near the boundaries unless more advanced wall handling is included in the method (see also the appendix, page 161). If the numerical method used immersed boundaries then "ghost" cells inside the wall could provide neighbour information to the boundary cells. Another option would be to use sub-cell meshes for boundary cells that require limiting, i.e. running a low order method on a refined mesh in the boundary cells to find a physically valid solution (Dumbser and Loubère 2016). No such special treatment of boundaries has been added in the slope limiter for the work presented in this thesis.

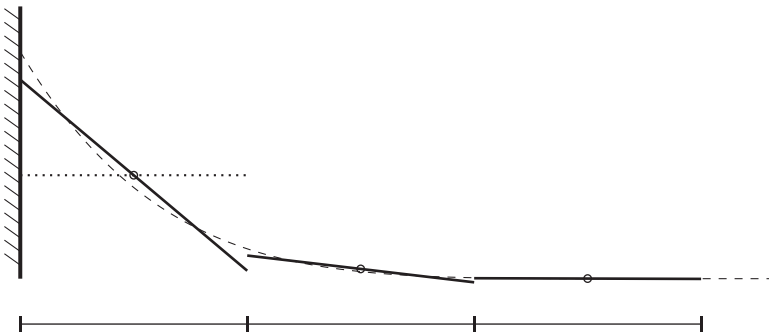


Figure 1.11: The first derivative of a function near a wall. The thick lines shows the linear DG approximation of the first derivative, and the dashed line shows the analytical first derivative. The dotted line shows the slope-limited solution near the wall which keeps the average first derivative, but removes the second derivative completely.

1.8 The volume-of-fluid method

In this thesis the two-phase air/water density field is approximated by a piecewise constant function space. The volume-of-fluid (VOF) method by Hirt and Nichols (1981) is used for evolving this density field in time while maintaining a sharp interface. In the VOF method a transport equation for the fluid density is solved for a normalised indicator function called the *colour function*, $c \in [0, 1]$, which is linearly related to the density. The true density field can be recovered from the indicator function c when the density of the two fluids are known, e.g.

$$\rho = c\,\rho_{\text{water}} + (1 - c)\rho_{\text{air}}. \tag{39}$$

Among the most commonly used VOF methods are the algebraic VOF schemes CICSAM (Ubbink 1997) and HRIC (Muzaferija et al. 1998). Geometric VOF schemes, where the interface is reconstructed by geometrical primitives in each cell, are also popular (Hirt and Nichols 1981; Roenby, Bredmose, and Jasak 2016; Youngs 1982), and mass-conserving level-set methods, which often couple a VOF method with the level-set equation to ensure mass conservation, are actively developed (Olsson and Kreiss 2005; Sussman and Puckett 2000; Touré, Fahsi, and Soulaïmani 2016).

In the following papers the algebraic VOF method HRIC has been used. In algebraic VOF methods the facet fluxes in the transport equation for the colour function are modified such that a donor cell on the free surface, e.g. the shaded cell in figure 1.12, must fully fill before it can transport any of the colour function to the cell above. This avoids diffusing the interface, which would happen if the upwind scheme was used and the donor cell colour 0.5 was transported across the top facet of the shaded cell with the facet velocity. An algebraic VOF scheme is nothing but a flux limiter (section 1.6) that minimises diffusion.

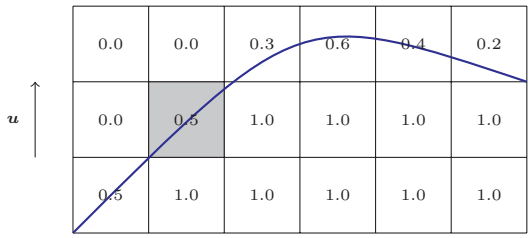


Figure 1.12: A colour-function field in the volume-of-fluid method.

High Resolution Interface Capturing—HRIC

The HRIC scheme by Muzaferija et al. (1998) calculates the flux of the colour function across a facet, c_F, based on a CBC-compliant scheme (see section 1.6). The downwind contribution to the facet flux is maximised, which contributes to sharpening the interface. As can be seen in figure 1.13, the HRIC scheme follows

the upper boundary of the TVD region in the normalised variable diagram,

$$\tilde{c}_F = \begin{cases} \tilde{c}_\mathrm{C} & \text{if} \quad \tilde{c}_\mathrm{C} \notin [0,1] \\ 2\tilde{c}_\mathrm{C} & \text{if} \quad \tilde{c}_\mathrm{C} \in [0, \frac{1}{2}] \\ 1 & \text{if} \quad \tilde{c}_\mathrm{C} \in [\frac{1}{2}, 1]. \end{cases} \tag{40}$$

Equation (40) is the same as the HYPER-C scheme by Leonard (1991) with Courant number equal to 0.5. The C in HYPER-C is for compressive, and Leonard shows that the heavily downwind-biased HYPER-C scheme will reduce all gradients to step functions. The VOF interface we seek is indeed a step function, but the problem with heavy sharpening is that it will align the free surface with the mesh. The true free surface is not, in general, aligned with the mesh. Too heavy sharpening causes stair-casing and unphysical wiggles of the interface (Ubbink 1997).

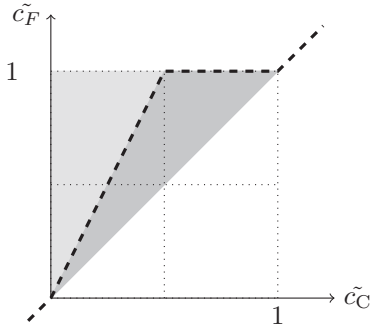


Figure 1.13: The HRIC scheme shown in the normalised-variable diagram.

In the HRIC scheme, the problem with downwind-sharpening causing alignment of the interface to the mesh is mitigated by allowing some diffusion when the angle θ between the facet normal $\boldsymbol{n}_F$ and the interface normal $\boldsymbol{n}_c$ is large. The diffusion is introduced by blending with the upwind value,

$$\tilde{c}_F^* = \gamma_{\boldsymbol{n}} \tilde{c}_F + (1 - \gamma_{\boldsymbol{n}}) \tilde{c}_\mathrm{C}, \qquad \gamma_{\boldsymbol{n}} = \sqrt{\cos(\theta)}. \tag{41}$$

Equations (40) and (41) may cause convergence problems if the Courant number Co is large. Even the original HYPER-C scheme—which is designed to be as close to the downwind scheme as possible while still being bounded—is less compressive than equation (40) for Courant numbers higher than 0.5. A Courant number dependency is therefore introduced,

$$\tilde{c}_F^{**} = \begin{cases} \tilde{c}_F^* & \text{if} \quad \mathrm{Co} < 0.3 \\ \tilde{c}_\mathrm{C} + (\tilde{c}_F^* - \tilde{c}_\mathrm{C}) \frac{0.7 - \mathrm{Co}}{0.7 - 0.3} & \text{if} \quad 0.3 < \mathrm{Co} < 0.7 \\ \tilde{c}_\mathrm{C} & \text{if} \quad \mathrm{Co} > 0.7 \end{cases} \tag{42}$$

Having calculated $\tilde{c}_F^{**}$, we can use equation (32) to calculate β and this can be used to formulate the DG FEM flux in equation (19) when assembling the transport equation for c.

One issue with using methods based on Leonard's normalised-variable diagram to calculate β is the dependency of $\tilde{c_{\mathrm{C}}}$ on c_{U}. On a general unstructured mesh the upstream cell value is not necessarily available. Having tested several methods for establishing an approximate upstream value, the method by Ubbink (1997) was found to give good results. In our HRIC implementation, the upstream value is calculated from

$$c_{\mathrm{U}} = c_{\mathrm{D}} - 2(\nabla c)_{\mathrm{C}} \cdot \boldsymbol{d}, \tag{43}$$

where $\boldsymbol{d}$ is the vector from the cell centre of K_{C} to the cell centre of K_{D}. We calculate $(\nabla c)_{\mathrm{C}}$, which is the gradient of the colour function in the cell centre of K_{C}, by a least-squares gradient-reconstruction procedure (Versteeg and Malalasekera 2007) which takes into account all cells that share one or more vertices with the centre cell.

The Courant number used in equation (42) in this work is the facet-based Courant number, which is computed as the facet average,

$$\mathrm{Co}_F = \operatorname*{avg}_F \left[\boldsymbol{u} \cdot \boldsymbol{n} \, \Delta t \, \frac{S_f}{V_c} \right], \tag{44}$$

where S_f is the facet area and V_c is the cell volume.

References

Akbas, M. et al. (2017). *An analogue of grad-div stabilization in nonconforming methods for incompressible flows*. report. Berlin : Weierstraß-Institut für Angewandte Analysis und Stochastik.

Alnæs, M. S. (2012). "UFL: a Finite Element Form Language". In: *Automated Solution of Differential Equations by the Finite Element Method, Volume 84 of Lecture Notes in Computational Science and Engineering*. Ed. by Logg, A., Mardal, K.-A., and Wells, G. N. Springer. Chap. 17.

Alnæs, M. S. et al. (2014). "Unified Form Language: A domain-specific language for weak formulations of partial differential equations". *ACM Transactions on Mathematical Software* 40.2.

Arnold, D. N. (1982). "An interior penalty finite element method with discontinuous elements". *SIAM journal on numerical analysis* 19.4, pp. 742–760.

Arnold, D. N. et al. (2002). "Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems". *SIAM J. Numer. Anal.* 39.5, pp. 1749–1779.

Babuška, I. and Dorr, M. R. (1981). "Error estimates for the combined h and p versions of the finite element method". *Numerische Mathematik* 37.2, pp. 257–277.

Baiges, J. et al. (2017). "An adaptive Fixed-Mesh ALE method for free surface flows". *Computer Methods in Applied Mechanics and Engineering* 313, pp. 159–188.

Baker, G. A., Jureidini, W. N., and Karakashian, O. A. (1990). "Piecewise Solenoidal Vector Fields and the Stokes Problem". *SIAM Journal on Numerical Analysis* 27.6, pp. 1466–1485.

Balay, S. et al. (2018). *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.9. Argonne National Laboratory.

Bangerth, W., Hartmann, R., and Kanschat, G. (2007). "deal.II—A general-purpose object-oriented finite element library". *ACM Transactions on Mathematical Software* 33.4.

Bassi, F. et al. (2006). "An artificial compressibility flux for the discontinuous Galerkin solution of the incompressible Navier–Stokes equations". *Journal of Computational Physics* 218.2, pp. 794–815.

Bassi, F. et al. (2007). "An implicit high-order discontinuous Galerkin method for steady and unsteady incompressible flows". *Computers & Fluids* 36.10, pp. 1529–1546.

Blasco, J., Codina, R., and Huerta, A. (1998). "A fractional-step method for the incompressible Navier–Stokes equations related to a predictor–multicorrector algorithm". *International Journal for Numerical Methods in Fluids* 28.10, pp. 1391–1419.

Bøckmann, A., Shipilova, O., and Skeie, G. (2012). "Incompressible SPH for free surface flows". *Computers & Fluids* 67, pp. 138–151.

Botti, L. and Di Pietro, D. A. (2011). "A pressure-correction scheme for convection-dominated incompressible flows with discontinuous velocity and continuous pressure". *Journal of Computational Physics* 3, pp. 572–585.

Braess, H. and Wriggers, P. (2000). "Arbitrary Lagrangian Eulerian finite element analysis of free surface flow". *Computer Methods in Applied Mechanics and Engineering* 190.1, pp. 95–109.

Brennen, C. and Whitney, A. K. (1970). "Unsteady, Free Surface Flows; Solutions Employing the Lagrangian Description of the Motion". In: *Hydrodynamics in the Ocean Environment; Eighth Symposium, Naval Hydrodynamics*. Ed. by Plesset, M. S., Wu, T. Y.-T., and Doroff, S. W. Arlington, VA, USA: Office of Naval Research, pp. 117–145.

Brooks, A. N. and Hughes, T. J. R. (1982). "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations". *Computer Methods in Applied Mechanics and Engineering* 32.1, pp. 199–259.

Cai, X. et al. (1998). "A Finite Element Method for Fully Nonlinear Water Waves". *Journal of Computational Physics* 143.2, pp. 544–568.

Carrero, J., Cockburn, B., and Schötzau, D. (2006). "Hybridized globally divergence-free LDG methods. Part I: The Stokes problem". *Mathematics of Computation* 75.254, pp. 533–563.

Cervone, A., Manservisi, S., and Scardovelli, R. (2010). "A FEM solver coupled to a multilevel VOF method for simulation of axisymmetric jets and to a front-tracking method for simulation of spreading droplets". *Atomization and Sprays* 20.2, pp. 115–131.

Cesmelioglu, A., Cockburn, B., and Qiu, W. (2017). "Analysis of a hybridizable discontinuous Galerkin method for the steady-state incompressible Navier-Stokes equations". *Mathematics of Computation* 86.306, pp. 1643–1670.

Chan, R. K.-C. and Street, R. L. (1970a). "A computer study of finite-amplitude water waves". *Journal of Computational Physics* 6.1, pp. 68–94.

— (1970b). *SUMMAC - A numerical model for water waves*. Tech. rep. TR-135. Stanford Univ. Dept. of Civil Engineering.

Chavent, G. and Salzano, G. (1982). "A finite-element method for the 1-D water flooding problem with gravity". *Journal of Computational Physics* 45.3, pp. 307–344.

Chavent, G. and Cockburn, B. (1989). "The local projection P0-P1-discontinuous-Galerkin finite element method for scalar conservation laws". *ESAIM: Mathematical Modelling and Numerical Analysis* 23.4, pp. 565–592.

Chessa, J. and Belytschko, T. (2003). "An extended finite element method for two-phase fluids". *Journal of Applied Mechanics* 70.1, pp. 10–17.

Cheung, S. W. et al. (2015). "Staggered discontinuous Galerkin methods for the incompressible Navier–Stokes equations". *Journal of Computational Physics* 302, pp. 251–266.

Chourushi, T. (2019). "Proposition of modified convection boundedness criterion and its evaluation for the development of bounded schemes". en. *Applied Mathematics and Computation* 346, pp. 710–739.

Ciarlet, P. G. (1976). *Numerical analysis of the finite element method*. English. Les Presses de L'Université de Montréal.

Cockburn, B. (2003). "Discontinuous Galerkin methods". *ZAMM* 83.11, pp. 731–754.

Cockburn, B. and Gopalakrishnan, J. (2005a). "Incompressible Finite Elements via Hybridization. Part I: The Stokes System in Two Space Dimensions". *SIAM Journal on Numerical Analysis* 43.4, pp. 1627–1650.

Cockburn, B. and Gopalakrishnan, J. (2005b). "Incompressible Finite Elements via Hybridization. Part II: The Stokes System in Three Space Dimensions". *SIAM Journal on Numerical Analysis* 43.4, pp. 1651–1672.

Cockburn, B., Gopalakrishnan, J., et al. (2010). "Analysis of HDG Methods for Stokes Flow". *Mathematics of computation* 80, pp. 723–760.

Cockburn, B., Kanschat, G., and Schötzau, D. (2004). "The local discontinuous Galerkin method for the Oseen equations". *Mathematics of Computation* 73.246, pp. 569–594.

— (2005). "A locally conservative LDG method for the incompressible Navier-Stokes equations". *Mathematics of Computation* 74.251, pp. 1067–1095.

— (2007). "A Note on Discontinuous Galerkin Divergence-free Solutions of the Navier–Stokes Equations". *Journal of Scientific Computing* 31.1/2, pp. 61–73.

Cockburn, B., Kanschat, G., Schötzau, D., and Schwab, C. (2002). "Local Discontinuous Galerkin Methods for the Stokes System". *SIAM Journal on Numerical Analysis* 40.1, pp. 319–343.

Cockburn, B. and Shu, C.-W. (1989). "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework". *Mathematics of Computation* 52.186, pp. 411–435.

Cockburn, B. and Shu, C.-W. (1991). "The Runge-Kutta local projection P1-discontinuous-Galerkin finite element method for scalar conservation laws". *Modélisation mathématique et analyse numérique* 25.3, pp. 337–361.

Di Pietro, D. A., Lo Forte, S., and Parolini, N. (2006). "Mass preserving finite element implementations of the level set method". *Applied Numerical Mathematics* 56.9, pp. 1179–1195.

Dumbser, M. and Loubère, R. (2016). "A simple robust and accurate a posteriori sub-cell finite volume limiter for the discontinuous Galerkin method on unstructured meshes". *Journal of Computational Physics* 319, pp. 163–199.

Eatock Taylor, R. (1996). "Analysis of Non-Linear Wave-Body Interactions Using Finite Elements". en. In: *Waves and Nonlinear Processes in Hydrodynamics*. Ed. by Grue, J., Gjevik, B., and Weber, J. E. Fluid Mechanics and Its Applications. Dordrecht: Springer Netherlands, pp. 51–62.

Egger, H. and Waluga, C. (2013). "*hp* analysis of a hybrid DG method for Stokes flow". *IMA Journal of Numerical Analysis* 33.2, pp. 687–721.

Einkemmer, L. and Wiesenberger, M. (2014). "A conservative discontinuous Galerkin scheme for the 2D incompressible Navier–Stokes equations". *Computer Physics Communications* 185.11, pp. 2865–2873.

Emamy, N. et al. (2017). "Implicit-explicit and explicit projection schemes for the unsteady incompressible Navier–Stokes equations using a high-order dG method". *Computers & Fluids* 154, pp. 285–295.

Enright, D. et al. (2002). "A Hybrid Particle Level Set Method for Improved Interface Capturing". *Journal of Computational Physics* 183.1, pp. 83–116.

Epshteyn, Y. and Rivière, B. (2007). "Estimation of penalty parameters for symmetric interior penalty Galerkin methods". *Journal of Computational and Applied Mathematics* 206.2, pp. 843–872.

Evans, J. A. (2011). "Divergence-free B-spline discretizations for viscous incompressible flows". thesis.

Fambri, F. (2019). "Discontinuous Galerkin Methods for Compressible and Incompressible Flows on Space–Time Adaptive Meshes: Toward a Novel Family of Efficient Numerical Methods for Fluid Dynamics". *Archives of Computational Methods in Engineering*, pp. 1–85.

Fambri, F. and Dumbser, M. (2017). "Semi-implicit discontinuous Galerkin methods for the incompressible Navier–Stokes equations on adaptive staggered Cartesian grids". *Computer Methods in Applied Mechanics and Engineering* 324, pp. 170–203.

Fehn, N., Munch, P., et al. (2019). "Hybrid multigrid methods for high-order discontinuous Galerkin discretizations". *arXiv:1910.01900 [physics]*.

Fehn, N., Wall, W. A., and Kronbichler, M. (2017). "On the stability of projection methods for the incompressible Navier–Stokes equations based on high-order discontinuous Galerkin discretizations". *Journal of Computational Physics* 351, pp. 392–421.

— (2018a). "Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows: High-performance discontinuous Galerkin for turbulent flows". *International Journal for Numerical Methods in Fluids* 88.1, pp. 32–54.

— (2018b). “Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows”. *Journal of Computational Physics* 372, pp. 667–693.

Ferrer, E. and Willden, R. H. J. (2011). “A high order Discontinuous Galerkin Finite Element solver for the incompressible Navier–Stokes equations”. *Computers & Fluids*. 10th ICFD Conference Series on Numerical Methods for Fluid Dynamics (ICFD 2010) 46.1, pp. 224–230.

Friedrich, O. (1998). “Weighted Essentially Non-Oscillatory Schemes for the Interpolation of Mean Values on Unstructured Grids”. *Journal of Computational Physics* 144.1, pp. 194–212.

Fu, G. (2019). “An explicit divergence-free DG method for incompressible flow”. *Computer Methods in Applied Mechanics and Engineering* 345, pp. 502–517.

Galvin, K. J. et al. (2012). “Stabilizing poor mass conservation in incompressible flow problems with large irrotational forcing and application to thermal convection”. *Computer Methods in Applied Mechanics and Engineering* 237-240, pp. 166–176.

Gaskell, P. H. and Lau, A. K. C. (1988). “Curvature-compensated convective transport: SMART, A new boundedness- preserving transport algorithm”. *International Journal for Numerical Methods in Fluids* 8.6, pp. 617–641.

Geuzaine, C. and Remacle, J.-F. (Sept. 10, 2009). “Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities”. *International Journal for Numerical Methods in Engineering* 79.11, pp. 1309–1331.

Giorgiani, G., Fernández-Méndez, S., and Huerta, A. (2014). “Hybridizable Discontinuous Galerkin with degree adaptivity for the incompressible Navier–Stokes equations”. *Computers & Fluids* 98, pp. 196–208.

Girault, V., Rivière, B., and Wheeler, M. (2005). “A discontinuous Galerkin method with nonoverlapping domain decomposition for the Stokes and Navier-Stokes problems”. *Mathematics of Computation* 74.249, pp. 53–84.

Girault, V., Rivière, B., and Wheeler, M. F. (2005). “A splitting method using discontinuous Galerkin for the transient incompressible Navier-Stokes equations”. *ESAIM: Mathematical Modelling and Numerical Analysis* 39.6, pp. 1115–1147.

Grooss, J. and Hesthaven, J. S. (2006). “A level set discontinuous Galerkin method for free surface flows”. *Computer Methods in Applied Mechanics and Engineering* 195.25, pp. 3406–3429.

Groß, S., Reichelt, V., and Reusken, A. (2006). “A finite element based level set method for two-phase incompressible flows”. *Computing and Visualization in Science* 9.4, pp. 239–257.

Groß, S. and Reusken, A. (2007). “An extended pressure finite element space for two-phase incompressible flows with surface tension”. *Journal of Computational Physics* 224.1, pp. 40–58.

Guermond, J.-L., Pasquetti, R., and Popov, B. (2011). “Entropy viscosity method for nonlinear conservation laws”. *Journal of Computational Physics*. Special issue High Order Methods for CFD Problems 230.11, pp. 4248–4267.

Guillén-González, F. and Tierra, G. (2012). "Superconvergence in velocity and pressure for the 3D time-dependent Navier-Stokes Equations". *SeMA Journal* 57.1, pp. 49–67.

Hansbo, P. and Larson, M. G. (2002). "Discontinuous Galerkin methods for incompressible and nearly incompressible elasticity by Nitsche's method". *Computer Methods in Applied Mechanics and Engineering* 191.17, pp. 1895–1908.

— (2008). "Piecewise divergence-free discontinuous Galerkin methods for Stokes flow". *Communications in Numerical Methods in Engineering* 24.5, pp. 355–366.

Harlow, F. H. and Welch, J. E. (1965). "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface". *The Physics of Fluids* 8.12, pp. 2182–2189.

Heimann, F. et al. (2013). "An unfitted interior penalty discontinuous Galerkin method for incompressible Navier–Stokes two-phase flow". *International Journal for Numerical Methods in Fluids* 71.3, pp. 269–293.

Heroux, M. A. and Willenbring, J. M. (2003). *Trilinos Users Guide*. Tech. rep. SAND2003-2952. Sandia National Laboratories.

Hesthaven, J. S. and Warburton, T. (2008). *Nodal Discontinuous Galerkin Methods*. Ed. by Marsden, J. E., Sirovich, L., and Antman, S. S. Vol. 54. Texts in Applied Mathematics. New York, NY: Springer New York.

Hindenlang, F. et al. (2012). "Explicit discontinuous Galerkin methods for unsteady problems". *Computers & Fluids* 61, pp. 86–93.

Hirt, C. W., Amsden, A. A., and Cook, J. L. (1974). "An arbitrary Lagrangian-Eulerian computing method for all flow speeds". *Journal of Computational Physics* 14.3, pp. 227–253.

Hirt, C. W., Cook, J. L., and Butler, T. D. (1970). "A Lagrangian method for calculating the dynamics of an incompressible fluid with free surface". *Journal of Computational Physics* 5.1, pp. 103–124.

Hirt, C. W. and Nichols, B. D. (1981). "Volume of fluid (VOF) method for the dynamics of free boundaries". *Journal of computational physics* 39.1, pp. 201–225.

Hu, C. and Shu, C.-W. (1999). "Weighted Essentially Non-oscillatory Schemes on Triangular Meshes". *Journal of Computational Physics* 150.1, pp. 97–127.

Huerta, A. and Liu, W. K. (1988). "Viscous flow with large free surface motion". *Computer Methods in Applied Mechanics and Engineering* 69.3, pp. 277–324.

Hughes, T. J. R. (1987). "Recent progress in the development and understanding of SUPG methods with special reference to the compressible Euler and Navier–Stokes equations". *International Journal for Numerical Methods in Fluids* 7.11, pp. 1261–1275.

Hughes, T. J. R., Liu, W. K., and Zimmermann, T. K. (Dec. 1981). "Lagrangian-Eulerian finite element formulation for incompressible viscous flows". *Computer Methods in Applied Mechanics and Engineering* 29.3, pp. 329–349.

Jasak, H., Weller, H., and Gosman, A. (1999). "High resolution NVD differencing scheme for arbitrarily unstructured meshes". *International Journal for Numerical Methods in Fluids* 31.2, pp. 431–449.

Jasak, H. (1996). “Error analysis and estimation for the finite volume method with applications to fluid flows”. PhD thesis. Imperial College, University of London.

Jeong, J. H. and Yang, D. Y. (1998). “Finite element analysis of transient fluid flow with free surface using VOF (volume-of-fluid) method and adaptive grid”. en. *International Journal for Numerical Methods in Fluids* 26.10, pp. 1127–1154.

Karakashian, O. and Katsaounis, T. (2006). “Numerical simulation of incompressible fluid flow using locally solenoidal elements”. *Computers & Mathematics with Applications* 51.9, pp. 1551–1570.

Karakashian, O. and Katsaounis, T. (2000). “A Discontinuous Galerkin Method for the Incompressible Navier-Stokes Equations”. In: *Discontinuous Galerkin Methods*. Ed. by Cockburn, B., Karniadakis, G. E., and Shu, C.-W. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer, pp. 157–166.

Karakus, A. et al. (2016). “A GPU-accelerated adaptive discontinuous Galerkin method for level set equation”. *International Journal of Computational Fluid Dynamics* 30.1, pp. 56–68.

Karniadakis, G. E., Israeli, M., and Orszag, S. A. (1991). “High-order splitting methods for the incompressible Navier-Stokes equations”. *Journal of Computational Physics* 97.2, pp. 414–443.

Kirby, R. C. and Logg, A. (2006). “A Compiler for Variational Forms”. *ACM Trans. Math. Softw.* 32.3, pp. 417–444.

Kirby, R. C., Logg, A., et al. (2012). “Common and unusual finite elements”. In: *Automated Solution of Differential Equations by the Finite Element Method*. Lecture Notes in Computational Science and Engineering. Springer, Berlin, Heidelberg, pp. 95–119.

Klaij, C. M., van der Vegt, J. J. W., and van der Ven, H. (2006). “Space–time discontinuous Galerkin method for the compressible Navier–Stokes equations”. *Journal of Computational Physics* 217.2, pp. 589–611.

Klein, B., Kummer, F., and Oberlack, M. (2013). “A SIMPLE based discontinuous Galerkin solver for steady incompressible flows”. *Journal of Computational Physics*, pp. 235–250.

Kopecz, S. (2012). *Ein gekoppeltes Finite-Elemente/Discontinuous-Galerkin-Verfahren zur Simulation von Strömungs-Transport-Problemen*. Kassel: Kassel Univ. Press.

Krank, B. et al. (2017). “A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow”. *Journal of Computational Physics* 348, pp. 634–659.

Kronbichler, M. and Wall, W. (2018). “A Performance Comparison of Continuous and Discontinuous Galerkin Methods with Fast Multigrid Solvers”. *SIAM Journal on Scientific Computing* 40.5, A3423–A3448.

Kronbichler, M. and Kormann, K. (2019). “Fast Matrix-Free Evaluation of Discontinuous Galerkin Finite Element Operators”. *ACM Trans. Math. Softw.* 45.3, 29:1–29:40.

Kuzmin, D. (Apr. 2010). "A vertex-based hierarchical slope limiter for p-adaptive discontinuous Galerkin methods". *Journal of Computational and Applied Mathematics*. Finite Element Methods in Engineering and Science (FEMTEC 2009) 233.12, pp. 3077–3085.

Labeur, R. J. and Wells, G. N. (2012). "Energy Stable and Momentum Conserving Hybrid Finite Element Method for the Incompressible Navier–Stokes Equations". *SIAM Journal on Scientific Computing* 34.2, A889–A913.

Lehrenfeld, C. and Schöberl, J. (2016). "High order exactly divergence-free Hybrid Discontinuous Galerkin Methods for unsteady incompressible flows". *Computer Methods in Applied Mechanics and Engineering*, pp. 339–361.

Leonard, B. P. (July 1979). "Adjusted quadratic upstream algorithms for transient incompressible convection". In: *4th Computational Fluid Dynamics Conference*. Williamsburg, VA, U.S.A: American Institute of Aeronautics and Astronautics, pp. 226–233.

— (1988). "Simple high-accuracy resolution program for convective modelling of discontinuities". *International Journal for Numerical Methods in Fluids* 8.10, pp. 1291–1318.

— (1991). "The ULTIMATE conservative difference scheme applied to unsteady one-dimensional advection". *Computer Methods in Applied Mechanics and Engineering* 88.1, pp. 17–74.

Liu, X.-D., Osher, S., and Chan, T. (1994). "Weighted Essentially Non-oscillatory Schemes". *Journal of Computational Physics* 115.1, pp. 200–212.

Liu, J.-G. and Shu, C.-W. (May 2000). "A High-Order Discontinuous Galerkin Method for 2D Incompressible Flows". *Journal of Computational Physics* 160.2, pp. 577–596.

Logg, A., Mardal, K.-A., and Wells, G. (2012). *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Springer Science & Business Media.

Logg, A., Ølgaard, K. B., et al. (2012). "FFC: the FEniCS form compiler". In: *Automated Solution of Differential Equations by the Finite Element Method*. Ed. by Logg, A., Mardal, K.-A., and Wells, G. Vol. 84. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 227–238.

Luo, H. et al. (2010). "A reconstructed discontinuous Galerkin method for the compressible Navier–Stokes equations on arbitrary grids". *Journal of Computational Physics* 229.19, pp. 6961–6978.

Ma, Q. W. and Yan, S. (2006). "Quasi ALE finite element method for nonlinear water waves". *Journal of Computational Physics* 212.1, pp. 52–72.

Marchandise, E. and Remacle, J.-F. (2006). "A stabilized finite element method using a discontinuous level set approach for solving two phase incompressible flows". *Journal of Computational Physics* 219.2, pp. 780–800.

Michoski, C. et al. (Jan. 2016). "A Comparison of Artificial Viscosity, Limiters, and Filters, for High Order Discontinuous Galerkin Solutions in Nonlinear Settings". en. *Journal of Scientific Computing* 66.1, pp. 406–434.

Moës, N., Dolbow, J., and Belytschko, T. (1999). "A finite element method for crack growth without remeshing". *International Journal for Numerical Methods in Engineering* 46.1, pp. 131–150.

Monaghan, J. J. (1994). "Simulating Free Surface Flows with SPH". *Journal of Computational Physics* 110.2, pp. 399–406.

Montlaur, A. et al. (2010). "Discontinuous Galerkin methods for the Navier–Stokes equations using solenoidal approximations". *International Journal for Numerical Methods in Fluids* 64.5, pp. 549–564.

Mozolevski, I., Süli, E., and Bösing, P. R. (2006). "Discontinuous Galerkin finite element approximation of the two-dimensional Navier-Stokes equations in stream-function formulation". *Communications in Numerical Methods in Engineering* 23.6, pp. 447–459.

Muzaferija, S. et al. (1998). "A Two-Fluid Navier-Stokes Solver to Simulate Water Entry". In: *Proceedings from the 22nd Symposium on Naval Hydrodynamics*. Washington, DC, pp. 277–289.

Nguyen, N. C., Peraire, J., and Cockburn, B. (2011). "An implicit high-order hybridizable discontinuous Galerkin method for the incompressible Navier–Stokes equations". *Journal of Computational Physics* 230, pp. 1147–1170.

Nichols, B. D. and Hirt, C. W. (1975). "Methods for calculating multi-dimensional, transient, free surface flows past bodies". In: *Proceedings of the First International Conference on Numerical Ship Hydrodynamics*. 20-22 Oct 1975. Gaithersburg, Maryland, USA, pp. 253–277.

Nickell, R. E., Tanner, R. I., and Caswell, B. (1974). "The solution of viscous incompressible jet and free-surface flows using finite-element methods". *Journal of Fluid Mechanics* 65.1, pp. 189–206.

Nitsche, J. A. (July 1971). "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* 36.1, pp. 9–15.

Noh, W. F. and Woodward, P. (1976). "SLIC (Simple Line Interface Calculation)". In: *Proceedings of the Fifth International Conference on Numerical Methods in Fluid Dynamics June 28 – July 2, 1976 Twente University, Enschede*. Ed. by Ehlers, J. et al. Vol. 59. Twente University, Enschede: Springer Berlin Heidelberg, pp. 330–340.

Ølgaard, K., Logg, A., and Wells, G. (2008). "Automated Code Generation for Discontinuous Galerkin Methods". *SIAM Journal on Scientific Computing* 31.2, pp. 849–864.

Ølgaard, K. B. and Wells, G. N. (2010). "Optimizations for quadrature representations of finite element tensors through automated code generation". *ACM Transactions on Mathematical Software* 37.1, pp. 1–23.

Olsson, E. and Kreiss, G. (Nov. 2005). "A conservative level set method for two phase flow". *Journal of Computational Physics* 210.1, pp. 225–246.

Osher, S. and Sethian, J. A. (1988). "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations". *Journal of Computational Physics* 79.1, pp. 12–49.

Owkes, M. and Desjardins, O. (2013). "A discontinuous Galerkin conservative level set scheme for interface capturing in multiphase flows". *Journal of Computational Physics* 249, pp. 275–302.

Pandare, A. K. and Luo, H. (2016). "A hybrid reconstructed discontinuous Galerkin and continuous Galerkin finite element method for incompressible flows on unstructured grids". *Journal of Computational Physics* 322, pp. 491–510.

Persson, P.-O. and Peraire, J. (2006). "An Efficient Low Memory Implicit DG Algorithm for Time Dependent Problems". In: *44th AIAA Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics.

Pesch, L. and van der Vegt, J. J. W. (2008). "A discontinuous Galerkin finite element discretization of the Euler equations for compressible and incompressible fluids". *Journal of Computational Physics* 227.11, pp. 5426–5446.

Peskin, C. S. (Jan. 2002). "The immersed boundary method". *Acta Numerica* 11, pp. 479–517.

Piatkowski, M., Müthing, S., and Bastian, P. (2018). "A stable and high-order accurate discontinuous Galerkin based splitting method for the incompressible Navier–Stokes equations". *Journal of Computational Physics* 356, pp. 220–239.

Ping-Li, H., Wen-Quan, T., and Mao-Zheng, Y. (2003). "Refinement of the convective boundedness criterion of Gaskell and Lau". *Engineering Computations* 20.8, pp. 1023–1043.

Pochet, F. et al. (2013). "A 3D strongly coupled implicit discontinuous Galerkin level set-based method for modeling two-phase flows". *Computers & Fluids* 87, pp. 144–155.

Popinet, S. (2003). "Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries". *Journal of Computational Physics* 190.2, pp. 572–600.

— (2014). *Basilisk*. www.basilisk.fr.

Qiu, W. and Shi, K. (2016). "A superconvergent HDG method for the incompressible Navier–Stokes equations on general polyhedral meshes". *IMA Journal of Numerical Analysis* 36.4, pp. 1943–1967.

Ramaswamy, B. and Kawahara, M. (1987). "Arbitrary Lagrangian-Eulerian finite element method for unsteady, convective, incompressible viscous free surface fluid flow". *International Journal for Numerical Methods in Fluids* 7.10, pp. 1053–1075.

Rhebergen, S., Cockburn, B., and van der Vegt, J. J. (Jan. 2013). "A space–time discontinuous Galerkin method for the incompressible Navier–Stokes equations". *Journal of Computational Physics* 233, pp. 339–358.

Rhebergen, S. and Wells, G. N. (2018). "A Hybridizable Discontinuous Galerkin Method for the Navier—Stokes Equations with Pointwise Divergence-Free Velocity Field". *Journal of Scientific Computing* 76.3, pp. 1484–1501.

Rivière, B. and Girault, V. (2006). "Discontinuous finite element methods for incompressible flows on subdomains with non-matching interfaces". *Computer Methods in Applied Mechanics and Engineering* 195.25, pp. 3274–3292.

Riviere, B. and Sardar, S. (2014). "Penalty-free discontinuous Galerkin methods for incompressible Navier–Stokes equations". *Mathematical Models and Methods in Applied Sciences* 24.06, pp. 1217–1236.

Robertson, I. and Sherwin, S. (1999). "Free-Surface Flow Simulation Using hp/Spectral Elements". *Journal of Computational Physics* 155.1, pp. 26–53.
Roenby, J., Bredmose, H., and Jasak, H. (2016). "A computational method for sharp interface advection". *Royal Society Open Science* 3.11, p. 160405.
Scardovelli, R. and Zaleski, S. (1999). "Direct Numerical Simulation of Free-Surface and Interfacial Flow". *Annual Review of Fluid Mechanics* 31.1, pp. 567–603.
Schötzau, D., Schwab, C., and Toselli, A. (2003a). "Stabilized hp-DGFEM for incompressible flow". *Mathematical Models and Methods in Applied Sciences* 13.10, pp. 1413–1436.
Schötzau, D., Schwab, C., and Toselli, A. (2003b). "Mixed hp-DGFEM for incompressible flows". *SIAM Journal on Numerical Analysis* 40.6, pp. 2171–2194.
Schroeder, P. W. and Lube, G. (2017). "Pressure-robust analysis of divergence-free and conforming FEM for evolutionary incompressible Navier–Stokes flows". *Journal of Numerical Mathematics* 25.4.
— (2018). "Divergence-Free H(div)-FEM for Time-Dependent Incompressible Flows with Applications to High Reynolds Number Vortex Dynamics". *Journal of Scientific Computing* 75.2, pp. 830–858.
Shahbazi, K., Fischer, P. F., and Ethier, C. R. (2007). "A high-order discontinuous Galerkin method for the unsteady incompressible Navier-Stokes equations". *Journal of Computational Physics* 222.1, pp. 391–407.
Shu, C.-W. (2003). "High-order Finite Difference and Finite Volume WENO Schemes and Discontinuous Galerkin Methods for CFD". *International Journal of Computational Fluid Dynamics* 17.2, pp. 107–118.
Steinmoeller, D. T., Stastna, M., and Lamb, K. G. (2013). "A short note on the discontinuous Galerkin discretization of the pressure projection operator in incompressible flow". *Journal of Computational Physics* 251, pp. 480–486.
Sun, Y.-w. and Kang, H.-g. (2012). "Application of CLEAR-VOF method to wave and flow simulations". *Water Science and Engineering* 5.1, p. 12.
Sussman, M. and Puckett, E. G. (2000). "A Coupled Level Set and Volume-of-Fluid Method for Computing 3D and Axisymmetric Incompressible Two-Phase Flows". *Journal of Computational Physics* 162.2, pp. 301–337.
Sussman, M., Smereka, P., and Osher, S. (1994). "A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow". *Journal of Computational Physics* 114.1, pp. 146–159.
Sweby, P. K. (1984). "High resolution schemes using flux limiters for hyperbolic conservation laws". *SIAM Journal on Numerical Analysis* 21.5, pp. 995–1011.
Tadmor, E. (1990). "Shock capturing by the spectral viscosity method". *Computer Methods in Applied Mechanics and Engineering* 80.1, pp. 197–208.
Tavelli, M. and Dumbser, M. (Sept. 2015). "A staggered space–time discontinuous Galerkin method for the incompressible Navier–Stokes equations on two-dimensional triangular meshes". *Computers & Fluids* 119, pp. 235–249.
— (Aug. 2016). "A staggered space–time discontinuous Galerkin method for the three-dimensional incompressible Navier–Stokes equations on unstructured tetrahedral meshes". en. *Journal of Computational Physics* 319, pp. 294–323.

Tornberg, A.-K. and Engquist, B. (2000). "Interface Tracking in Multiphase Flows". In: *Multifield Problems: State of the Art*. Ed. by Sändig, A.-M., Schiehlen, W., and Wendland, W. L. Berlin, Heidelberg: Springer, pp. 58–65.

Touré, M. K., Fahsi, A., and Soulaïmani, A. (Jan. 2016). "Stabilised finite-element methods for solving the level set equation with mass conservation". *International Journal of Computational Fluid Dynamics* 30.1, pp. 38–55.

Turek, S. (1999). *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Lecture Notes in Computational Science and Engineering. Berlin Heidelberg: Springer-Verlag.

Ubbink, O. (1997). "Numerical prediction of two fluid systems with sharp interfaces". PhD thesis. Imperial College, University of London.

van der Pijl, S. P. et al. (2005). "A mass-conserving Level-Set method for modelling of multi-phase flows". *International Journal for Numerical Methods in Fluids* 47.4, pp. 339–361.

van der Vegt, J. J. W. and van der Ven, H. (2002). "Space–Time Discontinuous Galerkin Finite Element Method with Dynamic Grid Motion for Inviscid Compressible Flows: I. General Formulation". *Journal of Computational Physics* 182.2, pp. 546–585.

van Leer, B. (1974). "Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second-order scheme". *Journal of computational physics* 14.4, pp. 361–370.

Versteeg, H. K. and Malalasekera, W. (2007). *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. 2nd ed. Prentice Hall.

VonNeumann, J. and Richtmyer, R. D. (1950). "A Method for the Numerical Calculation of Hydrodynamic Shocks". *Journal of Applied Physics* 21.3, pp. 232–237.

Weller, H. G. (2008). *A New Approach to VOF-based Interface Capturing Methods for Incompressible and Compressible flow*. Technical report. Reading, United Kingdom: OpenCFD.

Youngs, D. L. (1982). "Time-dependent multi-material flow with large fluid distortion". *Numerical methods for fluid dynamics*.

Yu, B. et al. (2001). "Discussion on Numerical Stability and Boundedness of Convective Discretized Scheme". *Numerical Heat Transfer, Part B: Fundamentals* 40.4, pp. 343–365.

Chapter 2

Summary of papers

Paper I discusses the challenges inherent in using a higher-order numerical approximation when simulating two-phase flows where the large difference in density between the heavy and the light fluid causes Gibbs oscillations in the resulting velocity field due to the sharp jump in momentum across the free surface. Two slope-limiting strategies for dealing with the instability at the free surface are presented. The investigation shows that the simplest presented method, where the *convected* velocity is slope limited by a scalar component-wise slope limiter while the *convecting* velocity is left unlimited, is able to completely remove the instability and produce exactly mass conserving results for 2D free surface flows.

Paper II shows how some standard pressure-correction schemes for solving the Navier–Stokes equations perform when used with a higher-order DG FEM discretisation. The ability of the schemes to maintain the exact incompressibility of a direct solver while allowing parallel solution is investigated and the convergence and efficiency of the schemes are presented.

Paper III builds on the work in the previous papers by extending the multi-phase solver to 3D and enabling full MPI parallelism through algebraic splitting of the Navier–Stokes saddle point block matrix system. This allows the DG FEM solution method to handle realistic physics with very complex free-surface behaviour. The presented numerical results compare well with published lab experiments.

Paper IV gives a condensed introduction to Ocellaris, a mass-conserving DG FEM solver for free-surface flows with sharp interfaces between the fluids. Ocellaris can simulate water entry and exit of objects in ocean waves with accurate capturing of the force on the object and the behaviour of the free surface. Ocellaris is implemented in Python and C++ with FEniCS as the backend for the mesh and finite element assembly. PETSc is used for solving the resulting linear systems. The source code and an extensive user guide is available at www.ocellaris.org along with demos and videos of what the solver is capable of. Ocellaris has been used to produce all numerical results shown in the included papers and is presented here by a short paper published in the Journal of Open Source Software.

Papers

Paper I

Slope limiting the velocity field in a discontinuous Galerkin divergence-free two-phase flow solver

Tormod Landet, Kent-Andre Mardal, Mikael Mortensen

Published in Computers & Fluids, DOI: 10.1016/j.compfluid.2019.104322

Abstract

Solving the Navier–Stokes equations when the density field contains a large and sharp discontinuity—such as a water/air free surface—is numerically challenging. Convective instabilities cause Gibbs oscillations which quickly destroy the solution. We investigate the use of slope limiters for the velocity field to overcome this problem in a way that does not compromise on the mass-conservation properties. The equations are discretised using the symmetric interior-penalty discontinuous Galerkin finite element method that is divergence-free to machine precision.

A slope limiter made specifically for exactly divergence-free (solenoidal) fields is presented and used to illustrate the difficulties in obtaining convectively stable fields that are also exactly solenoidal. The lessons learned from this are applied in constructing a simpler method based on the use of an existing scalar slope limiter applied to each velocity component.

We show by numerical examples how both presented slope limiting methods are vastly superior to the naive non-limited method. The methods can solve difficult two-phase problems with high density-ratios and high Reynolds numbers—typical for marine and offshore water/air simulations—in a way that conserves mass and stops unbounded energy growth caused by the Gibbs phenomenon.

I.1 Introduction

The incompressible and variable-density Navier–Stokes equations for the unknown velocity $\boldsymbol{u}$ and pressure p, with gravity $\boldsymbol{g}$ and spatially varying density ρ and viscosity μ,

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\right) = \nabla\cdot\mu\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right) - \nabla p + \rho\boldsymbol{g}, \tag{I.1}$$

$$\nabla\cdot\boldsymbol{u} = 0, \tag{I.2}$$

$$\frac{\partial \rho}{\partial t} + \boldsymbol{u}\cdot\nabla\rho = 0, \tag{I.3}$$

are used in applications where the fluid velocity is much smaller than the speed of sound and where gravity effects are important—such as for internal and surface gravity waves. Unfortunately, numerical problems will occur immediately when using equations (I.1) to (I.3) to study air/water free-surface physics with a higher-order spatial discretisation method. These problems are due to the sharp factor-1000 jump in momentum across the liquid/gas interface which causes Gibbs oscillations in the velocity field, even though the true velocity has no discontinuity at the interface. The energy in the velocity field will eventually blow up to destroy the solution if this non-linear convective instability is not handled with care.

The convergence of numerical approximations to the Navier–Stokes equations with variable and potentially discontinuous density and viscosity was studied by Liu and Walkington (2007). They show that if the continuous problem has a unique solution then a stable discontinuous Galerkin (DG) discretisation with a piecewise constant density field will converge to that solution. This work presents such a stable DG discretisation. Stable fractional step methods for solving the equations have been studied by Guermond and Quartapelle (2000), Guermond and Salgado (2009), and Pyo and Shen (2007). Our results are computed using a direct solver on the coupled velocity and pressure system in order to eliminate any fractional step splitting errors from influencing the conclusions.

In this work we will use a piecewise constant density field and the volume-of-fluid (VOF) method by Hirt and Nichols (1981) for evolving the density field in time while maintaining a sharp interface. In the VOF method, a transport equation for the density is solved for a normalised *colour function*, $c \in [0, 1]$, which is linearly related to the density. Among the most commonly used variations are the algebraic VOF schemes CICSAM (Ubbink 1997) and HRIC (Muzaferija et al. 1999). The level-set method (Osher and Sethian 1988) is another alternative for tracking free surfaces. Modern versions exist that significantly improve on the historical mass conservation problems of the level-set method—see, e.g., Olsson and Kreiss (2005) and Touré, Fahsi, and Soulaïmani (2016).

For most industrial applications, the above methods are implemented in solvers based on low-order finite volume methods where it is possible to ensure convective stability by appropriate use of flux limiters, see, e.g., the illustrative discussions and diagrams of Sweby (1984) and Leonard (1988). Flux limiters

based on TVD or ENO/WENO schemes are applied in the solution of the momentum equation, and some sort of stable interface sharpening scheme such as CICSAM or HRIC is used for the flux of density. Finite volume methods can be partially extended to higher-order by use of larger geometrical stencils, but this approach is not trivial to implement when using irregular grids, so in practice it is common that only immediate neighbours are used and most schemes are hence low-order.

For irregular geometries there are two major research directions that aim to enable higher-order methods: immersed boundary methods and finite element methods. Immersed boundary methods use regular background grids and perform special treatment of grid cells near or inside objects embedded in the computational domain (Peskin 2002). Finite element methods use basis functions with local support to enable high-order approximating functions on irregular meshes. The work in this paper is based on a discontinuous Galerkin (DG) finite element method (FEM). DG FEM has two main advantages over continuous Galerkin methods, the ease of using an upwind flux limiter for stabilising the linear convective instabilities, and the option to create exactly divergence-free and mass-conserving numerical schemes (Cockburn, Kanschat, and Schötzau 2004; Cockburn, Kanschat, and Schötzau 2005). There is one important drawback—the significantly increased number of degrees of freedom. We use a variation of the scheme from Cockburn, Kanschat, and Schötzau (2005), where instead of using a local discontinuous Galerkin, LDG, treatment of the elliptic term, a symmetric interior-penalty (SIP) treatment (Arnold 1982) is employed.

When using higher-order approximating polynomials, it is no longer sufficient to use only a flux limiter to obtain convective stability—a slope limiter must also be included in the method (Cockburn and Shu 1998; Cockburn and Shu 2001; Kuzmin 2010). The flux limiter ensures that the cell average values are bounded. When using higher-order basis functions, the solution can go out-of-bounds in localised regions due to steep gradients inside each cell. A slope limiter prevents this by flattening steep slopes near discontinuities while leaving the solution untouched near smooth maxima so that the method's high-order accuracy is retained. While a flux limiter is an implicit part of the equation system and has its stabilising effect included in the results from the linear equation solver, a slope limiter is applied to the resulting function as an explicit post-processing operator. Other options could be to use non-linear diffusion to combat the non-linear instability, or to apply spectral filtering, see, e.g., Michoski et al. 2016; Zingan et al. 2013.

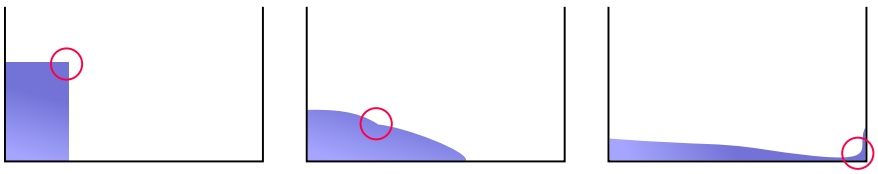
This paper starts with a description of the Gibbs instability and the discontinuous Galerkin method employed for solving the variable-density Navier–Stokes equations in sections I.2.1 to I.2.4. Slope limiting is then presented; first the hierarchical Taylor-based slope limiter by Kuzmin (2010) and Kuzmin (2013) in section I.3.1, and then the possibility of constructing a vector-field slope limiter that leaves the resulting velocity field both solenoidal and free from local maxima is explored in section I.3.2. After showing that it is likely not possible to obtain a single field that is both solenoidal and stable, separate limiting of the convecting and the convected velocity fields is introduced. Two alternatives are presented in

section I.3.3; both ensure solenoidal convecting velocities while keeping convective stability. Readers familiar with DG methods and slope limiting can start at section I.3.2, though the presentation builds directly on the preceding methods, so some referring back may be needed. Results from numerical tests are shown in section I.4 and discussion and concluding remarks can be found in sections I.5 and I.6.

I.2 The numerical method

I.2.1 Instabilities

The most common numerical instabilities related to handling of large density jumps are illustrated in figures I.1a to I.1c. A 'block' of water starts at rest in a box filled with air. Already in the very first time step (figure I.1a) the solution will start to blow up if one does not either apply smoothing to the density field, or stabilise the numerical scheme. Smoothing the initial field is not sufficient as the interface may pinch and become non smooth (figure I.1b), so continuous smoothing of the density field is necessary if this approach is selected. A level-set method is a natural way to implement a smoothed density field, see, e.g., Sussman, Smereka, and Osher (1994) and Unverdi and Tryggvason (1992). In this work we will not apply stabilisation through smoothing. Our aim is to decouple the stability of the method from the treatment of the density field.



(a) Initial condition (b) Pinching (c) Corner impact

Figure I.1: Illustration of typical numerical problems.

Figure I.1c illustrates a situation where the solenoidal properties of $\boldsymbol{u}$ are important. A divergence-free convecting velocity is required for the stability of the density transport equation (I.3), and any divergence quickly becomes a problem in difficult situations such as the corner impact where in our experience either mass loss or unbounded densities will occur if care is not taken to ensure that the convecting velocity used for density transport is solenoidal.

I.2.2 Preliminaries

The notation used in this paper is relatively standard. When looking at a facet between two cells (finite elements) we will denote one of the cells K^+ and the other K^- in an arbitrary, but repeatable manner. Function values in each cell will be given the same superscripts to distinguish between the values on opposite

sides of the facet in the discontinuous space. The average and jump operators across an internal facet are defined as

$$\{\{u\}\} = \frac{1}{2}(u^+ + u^-), \tag{I.4}$$

$$[\![u]\!] = u^+ - u^-, \tag{I.5}$$

$$[\![\boldsymbol{u}]\!] = \boldsymbol{u}^+ - \boldsymbol{u}^-, \tag{I.6}$$

$$[\![\boldsymbol{u}]\!]_{\boldsymbol{n}} = \boldsymbol{u}^+ \cdot \boldsymbol{n}^+ + \boldsymbol{u}^- \cdot \boldsymbol{n}^-. \tag{I.7}$$

Vector terms such as the velocity $\boldsymbol{u}$ and gravity $\boldsymbol{g}$ will be denoted with a bold font while scalars such as the pressure p, the density ρ, and the dynamic viscosity μ are shown in italics. Projection operators are written in blackboard bold. An example is the projection $\mathbb{D}$ into a solenoidal vector space. Sets are written in a calligraphic typeface. Fluxes of a quantity, used in DG facet integrals, are marked with a circumflex accent over the quantity, e.g., $\hat{\boldsymbol{u}}$, $\hat{p}$. An additional superscript is added when different definitions of the flux are used in different parts of the weak form, e.g., $\hat{\boldsymbol{u}}^{\boldsymbol{w}}$ and $\hat{\boldsymbol{u}}^p$, the flux of velocity related to convection and the flux of velocity related to the incompressibility constraint.

Let ∂K denote the boundary of element K. Γ_I is the inlet portion of the domain boundary $\partial\Omega$ where $\boldsymbol{u} \cdot \boldsymbol{n} < 0$ for an outwards pointing normal, $\boldsymbol{n}$. The set of all grid cells is $\mathcal{T}$ (the tessellation), and the set of all facets is $\mathcal{S}$ (the mesh skeleton). The set of outside facets is $\mathcal{S}_O = \mathcal{S} \cap \partial\Omega$ and the set of inside facets shared between two cells is $\mathcal{S}_I = \mathcal{S} \setminus \mathcal{S}_O$.

For a facet on the boundary, $F_b \in \mathcal{S}_O$, let the boundary cell be denoted K^+. We set terms related to the non-existent cell K^- to zero, so that $\boldsymbol{n}^+ \cdot [\![\boldsymbol{v}]\!] = \boldsymbol{n}^+ \cdot \boldsymbol{v}^+ = \boldsymbol{n} \cdot \boldsymbol{v}$ on F_b. The average is defined to be the K^+ value, $\{\{\boldsymbol{v}\}\} = \boldsymbol{v}^+ = \boldsymbol{v}$ on $\mathcal{S}_O$. In both cases the superscript "+" can be dropped as there is no ambiguity.

Let $P_k(K)$ denote the space of polynomials of-order k on an element and $P_k(F)$ denote the space of polynomials of order k on a facet. The basis functions in $P_k(K)$ are discontinuous across facets and functions in $P_k(F)$ are discontinuous across edges (vertices in 2D). Let the superscript $\bullet^{n+1}$ denote a value at time step $t = \Delta t(n+1)$. For nabla the conventions $(\nabla \boldsymbol{u})_{ij} = \partial_j \boldsymbol{u}_i$ and $(\nabla \cdot \boldsymbol{\sigma})_i = \partial_j \sigma_{ij}$ are used.

I.2.3 Discretisation

We will approximate the unknown functions in space by using discontinuous Lagrange polynomial function spaces with polynomial order $k = 2$ for the velocity in $d = 2$ spatial dimensions. Let the Galerkin test functions for $\{\boldsymbol{u}, p, \rho\}$ be denoted respectively $\{\boldsymbol{v}, q, r\}$. The discontinuous function spaces are not restricted at the boundaries—all boundary conditions will be imposed weakly—so the trial and test functions share spaces,

$$\begin{aligned} \boldsymbol{u}, \boldsymbol{v} &\in [P_k(K)]^d, \\ p, q &\in P_{k-1}(K), \\ \rho, r &\in P_0(K). \end{aligned} \tag{I.8}$$

I.2.3.1 Two-phase density transport

The transport equation (I.3) for the density is modified by the introduction of a solenoidal convecting velocity field $\boldsymbol{w}$ which is close to $\boldsymbol{u}$, but may not be identical as will be explained in sections I.2.4 and I.3.3. Boundary conditions are needed only on the inlet. The strong form can then be written

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \boldsymbol{w} \cdot \nabla \rho = 0 \qquad & \text{in } \Omega, \\ \rho = \rho_I \qquad & \text{on } \Gamma_I. \end{aligned} \tag{I.9}$$

In VOF methods the fluid properties are expressed in terms of an indicator function, $c \in [0, 1]$. The true density and viscosity fields can easily be recovered from c when the density and kinematic viscosity properties of the two fluids are known,

$$\rho = c\rho_{\text{water}} + (1 - c)\rho_{\text{air}}, \tag{I.10}$$

$$\mu = [c\nu_{\text{water}} + (1 - c)\nu_{\text{air}}]\, \rho. \tag{I.11}$$

In order to compute c, the transport equation (I.9) for the density is modified by inserting equation (I.10). Now the solution $c^{n+1} \in P_0(K)$ can be found by expressing the resulting equation on weak form and integrating by parts,

$$\begin{aligned} &\int_{\mathcal{T}} \frac{1}{\Delta t}(\gamma_1 c^{n+1} + \gamma_2 c^n + \gamma_3 c^{n-1}) r \, \mathrm{d}\boldsymbol{x} \\ &\quad - \int_{\mathcal{T}} c^{n+1} \boldsymbol{w} \cdot \nabla r \, \mathrm{d}\boldsymbol{x} + \int_{\mathcal{S}} \hat{c}^{n+1} \, \boldsymbol{w} \cdot \boldsymbol{n}^+ \, [\![\, r \,]\!] \, \mathrm{d}s = 0, \end{aligned} \tag{I.12}$$

where we have assumed that the convecting velocity $\boldsymbol{w}$ is $\mathrm{H}^{\mathrm{div}}$-conforming such that the flux is continuous across facets, $[\![\, \boldsymbol{w} \,]\!]_{\boldsymbol{n}} = 0$.

A second-order backwards-differencing formulation, BDF2, is used for time integration. The parameters are $\{\gamma_1, \gamma_2, \gamma_3\} = \{3/2, -2, 1/2\}$. The BDF2 method is monotonicity-preserving when started using a backward Euler step (Hundsdorfer, Ruuth, and Spiteri 2003), though the time step required to preserve monotonicity is half that of backward Euler. The benefit is that the method is second-order—and small time steps are anyhow required to keep the interface sharp (Muzaferija et al. 1999). Second-order extrapolation is used for the convective velocity,

$$\boldsymbol{w} = 2\boldsymbol{w}^n - \boldsymbol{w}^{n-1}, \tag{I.13}$$

which makes ρ^{n+1} independent of the computed velocity at time $t = (n + 1)\Delta t$. The density transport equation is hence uncoupled from the momentum equation.

For the density flux, $\hat{c}^{n+1}$, the most stable choice is to take the upwind value, which means that the boundary condition $\hat{\rho} = \rho_I$ is used to determine $\hat{c}^{n+1}$ on inlet facets, $\mathcal{S} \cap \Gamma_I$. For the internal facets the term related to the flux can be written on upwind form as

$$\hat{c}^{n+1} \, \boldsymbol{w} \cdot \boldsymbol{n}^+ = \left[\!\!\left[c^{n+1} \frac{1}{2}(\boldsymbol{w} \cdot \boldsymbol{n} + |\boldsymbol{w} \cdot \boldsymbol{n}|) \right]\!\!\right], \tag{I.14}$$

and, by replacing the plus by a minus in equation (I.14), the downwind flux can similarly be computed. Using the upwind and downwind fluxes on each facet, a downwind-blended compressive interface flux can be applied to the density transport such as CICSAM (Ubbink 1997) or HRIC (Muzaferija et al. 1999). Such blended fluxes sharpen the interface between the two fluid layers, but remain convectively stable unlike downwind or central fluxes. Both CICSAM and HRIC are algebraic VOF methods which define facet-wise blending factors to combine the upwind and downwind fluxes into one linearly stable flux. The results in section I.4 are computed using the HRIC method.

One important note is that standard VOF flux limiters from finite volume methods ensure that the c field remains sharp and bounded based on a convecting velocity field that is piecewise constant on each facet. Such a field can easily be computed from $\boldsymbol{w}$ and as long as $\boldsymbol{w}$ is solenoidal, so is the piecewise constant field. It is this field that is used when solving for c^{n+1}, since $\boldsymbol{w}$ is only needed on the facets. For $r \in P_0(K)$, the volume integral term containing $\boldsymbol{w}$ in equation (I.12) is identically zero.

I.2.3.2 The variable-density Navier–Stokes equations

The strong form of the variable-density Navier–Stokes equations, where the convecting velocity is replaced by $\boldsymbol{w}$, can be written

$$\begin{aligned}
\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{w}\cdot\nabla)\boldsymbol{u}\right) &= \nabla\cdot\mu(\nabla\boldsymbol{u} + \left(\nabla\boldsymbol{u}\right)^T) - \nabla p + \rho\boldsymbol{g} && \text{in } \Omega, \quad \text{(I.15)}\\
\nabla\cdot\boldsymbol{u} &= 0 && \text{in } \Omega,\\
\boldsymbol{u} &= \boldsymbol{u}_D && \text{on } \Gamma_D,\\
\frac{\partial \boldsymbol{u}}{\partial n} &= \boldsymbol{a} && \text{on } \Gamma_N,
\end{aligned}$$

where Γ_D and Γ_N are the parts of the boundary where Dirichlet and Neumann boundary conditions are applied respectively; $\Gamma_N = \partial\Omega \setminus \Gamma_D$. The imposed Dirichlet boundary value is $\boldsymbol{u}_D$, while $\boldsymbol{a}$ is the imposed gradient of the velocity in the direction of the normal on the Neumann part of the boundary. Dirichlet boundary conditions can be enforced on external facets due to the elliptic viscosity term, $\nabla\cdot\mu(\nabla\boldsymbol{u} + \left(\nabla\boldsymbol{u}\right)^T)$. The Navier–Stokes equations will be written on weak form, see equation (I.20), but prior to that we will briefly discuss the symmetric interior-penalty (SIP) method. First, the boundary condition, $\boldsymbol{u} = \boldsymbol{u}_D$, is written on weak form on an external facet F,

$$\int_F \boldsymbol{u}\cdot\boldsymbol{v}\,\mathrm{d}s = \int_F \boldsymbol{u}_D\cdot\boldsymbol{v}\,\mathrm{d}s, \tag{I.16}$$

and then—using the stabilisation scheme proposed by Nitsche (1971)—the test function $\boldsymbol{v}$ is replaced by a Petrov-Galerkin test function $\tilde{\boldsymbol{v}} = \kappa_\mu\boldsymbol{v} - \mu\left(\nabla\boldsymbol{v} + (\nabla\boldsymbol{v})^T\right)\cdot\boldsymbol{n}$. This method is extended to enforce continuity across

internal facets which is necessary for stability (Arnold 1982). This gives

$$\int_F \kappa_\mu \left[\!\left[\boldsymbol{u} \right]\!\right] \cdot \left[\!\left[\boldsymbol{v} \right]\!\right] \mathrm{d}s - \int_F (\{\{\mu \left(\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T\right)\}\} \cdot \boldsymbol{n}^+) \cdot \left[\!\left[\boldsymbol{u} \right]\!\right] \mathrm{d}s = 0, \tag{I.17}$$

where κ_μ is a penalty parameter which must be sufficiently large to ensure stability. The analyses in Epshteyn and Rivière (2007) and Shahbazi, Fischer, and Ethier (2007) guide us in defining the penalty parameter as a function of the minimum and maximum dynamic viscosities, $\mu_{\min}$ and $\mu_{\max}$, the order k of the approximating polynomials, and the surface area S_K and volume V_K of each cell K,

$$\kappa_\mu = 3\,\frac{\mu_{\max}^2}{\mu_{\min}}\,k(k+1)\,\max_K\left(\frac{S_K}{V_K}\right). \tag{I.18}$$

The same scheme is used for the left-hand side of the momentum equation as for the density transport equation, but a pure upwind flux is used without any downwind blending for the convective term. To avoid overloading the notation we now drop the $\bullet^{n+1}$ superscript on the unknown quantities. The upwind flux related to convection can then be written

$$\hat{\boldsymbol{u}}^{\boldsymbol{w}}\,\boldsymbol{w}\cdot\boldsymbol{n}^+ = \left[\!\!\left[\boldsymbol{u}\frac{1}{2}(\boldsymbol{w}\cdot\boldsymbol{n} + |\boldsymbol{w}\cdot\boldsymbol{n}|) \right]\!\!\right]. \tag{I.19}$$

Both the pressure gradient and the viscosity on the right-hand side of the momentum equation are integrated by parts. The resulting weak form is a direct combination of the LDG Navier–Stokes method by Cockburn, Kanschat, and Schötzau (2005) and the SIP diffusion method by Arnold (1982). Both these references contain more details and proofs of stability. The stability of the convective and the diffusive terms are not interconnected, so replacing the LDG elliptic operator with the SIP version is unproblematic. Using SIP in this way is not novel, see e.g. Cockburn, Kanschat, and Schötzau (2007) and Shahbazi, Fischer, and Ethier (2007). The treatment of the momentum transport and the inter-cell continuity, both described above, are easily recognisable in the complete weak form,

$$\begin{aligned}
&\int_{\mathcal{T}} \frac{\rho}{\Delta t}(\gamma_1\boldsymbol{u} + \gamma_2\boldsymbol{u}^n + \gamma_3\boldsymbol{u}^{n-1})\boldsymbol{v}\,\mathrm{d}\boldsymbol{x} \\
&\quad - \int_{\mathcal{T}} \boldsymbol{u}\cdot\nabla\cdot(\rho\boldsymbol{v}\otimes\boldsymbol{w})\,\mathrm{d}\boldsymbol{x} \;+\; \int_{\mathcal{S}} \boldsymbol{w}\cdot\boldsymbol{n}^+\,\hat{\boldsymbol{u}}^{\boldsymbol{w}}\cdot\left[\!\left[\rho\boldsymbol{v} \right]\!\right]\mathrm{d}s \\
&\quad + \int_{\mathcal{T}} \mu\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right) : \nabla\boldsymbol{v}\,\mathrm{d}\boldsymbol{x} \;+\; \int_{\mathcal{S}_I} \kappa_\mu \left[\!\left[\boldsymbol{u} \right]\!\right]\cdot\left[\!\left[\boldsymbol{v} \right]\!\right]\mathrm{d}s \\
&\quad - \int_{\mathcal{S}} (\{\{\mu\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right)\}\}\cdot\boldsymbol{n}^+)\cdot\left[\!\left[\boldsymbol{v} \right]\!\right]\mathrm{d}s \\
&\quad - \int_{\mathcal{S}_I} (\{\{\mu\left(\nabla\boldsymbol{v} + (\nabla\boldsymbol{v})^T\right)\}\}\cdot\boldsymbol{n}^+)\cdot\left[\!\left[\boldsymbol{u} \right]\!\right]\mathrm{d}s \\
&\quad - \int_{\mathcal{T}} p\,\nabla\cdot\boldsymbol{v}\,\mathrm{d}\boldsymbol{x} \;+\; \int_{\mathcal{S}} \hat{p}\,\boldsymbol{n}^+\cdot\left[\!\left[\boldsymbol{v} \right]\!\right]\mathrm{d}s \;=\; \int_{\mathcal{T}} \rho\,\boldsymbol{g}\,\mathrm{d}\boldsymbol{x}.
\end{aligned} \tag{I.20}$$

where the flux of pressure is taken as $\hat{p} = \{\{p\}\}$.

The continuity equation (I.2) is also integrated by parts using $\hat{\boldsymbol{u}}^p = \{\{\boldsymbol{u}\}\}$ as the flux related to the incompressibility constraint,

$$\int_{\mathcal{S}} \hat{\boldsymbol{u}}^p \cdot \boldsymbol{n}^+ [\![q]\!] \, \mathrm{d}s \; - \; \int_{\mathcal{T}} \boldsymbol{u} \cdot \nabla q \, \mathrm{d}\boldsymbol{x} \; = \; 0. \tag{I.21}$$

Dirichlet boundaries On the inflow part of the Dirichlet boundary, take $\hat{\boldsymbol{u}}^{\boldsymbol{w}} = \boldsymbol{u}_D$, and on the outflow part take $\hat{\boldsymbol{u}}^{\boldsymbol{w}} = \boldsymbol{u}$, i.e., the upwind values are used. On the whole Dirichlet boundary let $\hat{\boldsymbol{u}}^p = \boldsymbol{u}_D$ and $\hat{p} = p$. The viscous penalty and symmetrisation terms from equation (I.17) can be used also on the domain boundary by replacing $\boldsymbol{u}^-$ by $\boldsymbol{u}_D$. As noted in the references used to define κ_μ above, the best choice is to use twice the amount of penalisation on external facets compared to the interior facets. This leads to external boundary integrals

$$\int_{\Gamma_D} 2\kappa_\mu (\boldsymbol{u} - \boldsymbol{u}_D) \cdot \boldsymbol{v} \, \mathrm{d}s - \int_{\Gamma_D} (\mu \nabla \boldsymbol{v} \cdot \boldsymbol{n}) \cdot (\boldsymbol{u} - \boldsymbol{u}_D) \, \mathrm{d}s = 0. \tag{I.22}$$

Neumann boundaries For the Neumann boundaries we take $\hat{\boldsymbol{u}}^{\boldsymbol{w}} = \boldsymbol{u}$, $\hat{\boldsymbol{u}}^p = \boldsymbol{u}$ and $\hat{p} = p$. The extra viscous terms for symmetrisation and penalty are removed and only the normal integration by parts terms are left. The surface term becomes

$$- \int_{\Gamma_N} \mu \, \boldsymbol{a} \cdot \boldsymbol{v} \, \mathrm{d}s. \tag{I.23}$$

Pure Neumann boundary conditions will not be used, but we will use Dirichlet for one velocity component and Neumann for the other to implement free-slip boundary conditions on planes that are parallel to the axes. The definitions above can easily be split into component-wise treatment of boundary conditions.

Solution algorithm

The Navier–Stokes equations and the density transport equation are solved in a decoupled manner for each time step in the following order:

1. Compute an explicit convecting velocity $\boldsymbol{w}^{n+1}$ by use of equation (I.13). For the first time step take $\boldsymbol{w}^1 = \boldsymbol{u}^0$, unless both $\boldsymbol{w}^0 = \boldsymbol{u}^0$ and $\boldsymbol{w}^{-1} = \boldsymbol{u}^{-1}$ are given as input.

2. Find $c^{n+1} \in P_0(K)$ such that equation (I.12) is satisfied.

3. Compute the density ρ^{n+1} and viscosity μ^{n+1} by use of equations (I.10) and (I.11).

4. Use the computed coefficient fields $\boldsymbol{w}^{n+1}$, μ^{n+1} and ρ^{n+1} to find $\boldsymbol{u}^{n+1} \in [P_k(K)]^d$ and $p^{n+1} \in P_{k-1}(K)$ from equations (I.20) and (I.21).

I.2.4 $\mathrm{H}^{\mathrm{div}}$ projection of the velocity field

In finite element methods for solving the Navier–Stokes equations it is common to impose the incompressibility criterion, $\nabla \cdot \boldsymbol{u} = 0$, weakly by multiplying with a scalar test function, q, and integrating over the domain. This term, $\int_\Omega \nabla \cdot \boldsymbol{u}\, q \,\mathrm{d}\boldsymbol{x}$, will appear directly in a coupled solver and as the right-hand side in the Poisson equation for the pressure in a pressure-correction fractional-step scheme such as the commonly used incremental pressure-correction scheme (IPCS).

Imposing the incompressibility criterion weakly in the space of the pressure is sufficient for stability, but one must require exact incompressibility to locally conserve mass and momentum. Below we explain the core ideas of the method presented in Cockburn, Kanschat, and Schötzau (2005). By using this method, the resulting divergence will be zero almost to machine precision, approximately 10^{-13} on each cell in our tests. We calculate the cell-wise error by computing the integrated absolute value of the divergence internally in each cell, $\int_K |\nabla \cdot \boldsymbol{u}| \,\mathrm{d}\boldsymbol{x}$, and add to it the error in flux continuity between cells on each connected facet, $\int_{\partial K} |\, [\![\, \boldsymbol{u} \cdot \boldsymbol{n} \,]\!] \,| \,\mathrm{d}s$.

Let us start by noting that $\mathrm{H}^{\mathrm{div}}$-conforming finite elements exist, and that such elements of a given polynomial order will be subspaces of the fully discontinuous elements of the same order with the same cell geometry. Such elements impose continuity of normal fluxes across facets and hence have fewer global degrees of freedom. We follow Cockburn, Kanschat, and Schötzau (2005) and define a projection from our fully discontinuous velocity $\boldsymbol{u}$ into a velocity $\boldsymbol{w} = \mathbb{P}\boldsymbol{u}$ that exist in a space of polynomials that are consistent with the $\mathrm{H}^{\mathrm{div}}$-conforming elements.

The projection operator $\boldsymbol{w} = \mathbb{P}\boldsymbol{u}$ is defined in a cell-wise manner. This local projection is hence very fast and consists of finding $\boldsymbol{w} \in [P_k(K)]^d$ such that

$$\int_F \boldsymbol{w} \cdot \boldsymbol{n}\, v_1 \,\mathrm{d}s = \int_F \hat{\boldsymbol{u}}^p \cdot \boldsymbol{n}\, v_1 \,\mathrm{d}s \qquad \forall v_1 \in P_k(F), F \in \partial K, \tag{I.24}$$

$$\int_K \boldsymbol{w} \cdot \boldsymbol{v}_2 \,\mathrm{d}x = \int_K \boldsymbol{u} \cdot \boldsymbol{v}_2 \,\mathrm{d}x \qquad \forall \boldsymbol{v}_2 \in \boldsymbol{N}_{k-1}(K). \tag{I.25}$$

The first equation ensures the continuity of the normal velocities across each facet. The continuity stems from using a single valued flux, which is here $\hat{\boldsymbol{u}}^p = \{\{\boldsymbol{u}\}\}$ on internal facets and $\hat{\boldsymbol{u}}^p = \boldsymbol{u}_D$ on external facets ($\hat{\boldsymbol{u}}^p = \boldsymbol{u}$ on Γ_N). This flux is consistent for continuous velocity fields, as long as the solution has sufficient smoothness to be well represented by the numerical discretisation.

In equation (I.25), the space $\boldsymbol{N}_{k-1}(K)$ is the Nédélec $H(\mathrm{curl})$ element of the first kind of order $k-1$, see, e.g., Kirby et al. (2012) and Nédélec (1986). The dimension of the Brezzi-Douglas-Marini (BDM) element $P_k(F) \times \boldsymbol{N}_{k-1}(K)$ is the same as that of the Discontinuous Lagrange DGk element on each cell (Brezzi and Fortin 1991). It is hence possible to form square projection matrices between the two spaces in each cell. The projection $\mathbb{P}\boldsymbol{u}$ is not square globally since the degrees of freedom related to v_1 are shared between exactly two cells on all internal facets. This leads to the test space having fewer global degrees of freedom than the fully discontinuous trial space.

The properties of this BDM-like projection used as a velocity post-processing step in a discontinuous Galerkin method is given by Cockburn, Kanschat, and Schötzau (2005). The projection gives a continuous flux $\boldsymbol{w} \cdot \boldsymbol{n}$ on all inter-element facets, and also ensures that the total flux across each individual cell's facets is zero. Fulfilling these two criteria is what we mean by exact incompressibility.

I.3 Slope limiting

The numerical diffusion due to upwinding in the spatial DG scheme is sufficient to stabilise the convective operator, and avoid spurious oscillations, when piecewise constant approximating functions are employed. The reason is that a pure upwind flux is a stable flux limiter—and hence ensures the boundedness of the cell averages—which is sufficient for stability in first-order methods where the cell averages are the only degrees of freedom. A first-order flux limiter can be used to obtain higher-order convergence, but then it must be combined with higher-order approximating functions inside each cell. This is the approach taken in this work. The convective operators are then additionally stabilised through the use of a slope limiter, where the oscillations inside each element are handled in an element-by-element post-processing procedure (Cockburn and Shu 1998), while the cell-average values are still bounded due to the flux limiter.

I.3.1 The hierarchical Taylor-polynomial-based slope limiter

We base our slope-limiting methods on the hierarchical vertex-based slope limiter for scalar fields by Kuzmin (2010) and Kuzmin (2013). Kuzmin's slope-limiting procedure is based on using discontinuous Taylor-polynomial function spaces. The first step is hence to project a scalar function ϕ in the discontinuous Lagrange function space—which is what is used in the rest of our numerical scheme—to a function ϕ_t in the discontinuous Taylor function space. This projection $\phi_t = \mathbb{T}\phi$ is local to each cell, and can be applied and inverted exactly by a single matrix-vector product in each cell, so the cost of converting back and forth is negligible.

The discontinuous Taylor function space applied in Kuzmin's method is slightly altered from the standard definition by using the cell-average instead of the cell-centre value for the constant term in the polynomial expansion for each cell. This allows working separately with the conserved quantity—the average value in each cell which is known to be bounded due to the flux limiter—and the slopes of the function in the cell, which now do not influence the conservation

properties. The expansion

$$\begin{aligned}\phi_t(x,y) = \bar{\phi} &+ \left.\frac{\partial\phi}{\partial x}\right|_c \alpha_1(x-x_c) + \left.\frac{\partial\phi}{\partial y}\right|_c \alpha_1(y-y_c) \\ &+ \left.\frac{\partial^2\phi}{\partial x^2}\right|_c \alpha_2 \left[\frac{(x-x_c)^2}{2} - \overline{\frac{(x-x_c)^2}{2}}\right] \\ &+ \left.\frac{\partial^2\phi}{\partial y^2}\right|_c \alpha_2 \left[\frac{(y-y_c)^2}{2} - \overline{\frac{(y-y_c)^2}{2}}\right] \\ &+ \left.\frac{\partial^2\phi}{\partial x\partial y}\right|_c \alpha_2 \left[\frac{(x-x_c)(y-y_c)}{2} - \overline{\frac{(x-x_c)(y-y_c)}{2}}\right]\end{aligned} \tag{I.26}$$

is used to describe a second-order polynomial function on a triangle where the six coefficients are $\{\bar{\phi}, {}^{\partial\phi}/_{\partial x}|_c, \cdots, {}^{\partial^2\phi}/_{\partial x\partial y}|_c\}$. The number of coefficients is the same as the number of nodes in a second-order Lagrange-polynomial function space on a triangle, which ensures that the transformation matrix resulting from $\mathbb{T}$ is square on each cell. The same is also true for tetrahedra and for higher and lower polynomial orders, the number of degrees of freedom in the discontinuous Taylor and the discontinuous Lagrange function spaces are the same. Over-lined terms in equation (I.26), such as $\bar{\phi}$ and $\overline{(x-x_c)^2}/2$, denote cell averages, and the restriction $\cdot|_c$ signifies evaluation in the cell centre, x_c, taken as the geometric midpoint of the cell. Slope limiter coefficients $\alpha_i \in [0,1]$ are multiplied with the i'th derivative terms, using the same α-factors for all derivatives of the same order.

We follow Kuzmin (2013) and start by defining a set of bi-linear functions on each cell: linear reconstructions of the function and its derivatives. Due to their linear nature, they will have their maxima and minima at the vertices, and this property is used to determine α_i. For polynomial basis functions of order k, the derivatives up to order $k-1$ should be approximated in this manner. For a scalar function $\phi_t \in P_2(K)$ on a triangle there are three bi-linear reconstructions,

$$\tilde{\phi}_0(x,y) = \bar{\phi} + \tilde{\alpha}_1 \left[\left.\frac{\partial\phi}{\partial x}\right|_c (x-x_c) + \left.\frac{\partial\phi}{\partial y}\right|_c (y-y_c)\right], \tag{I.27}$$

$$\tilde{\phi}_x(x,y) = \left.\frac{\partial\phi}{\partial x}\right|_c + \tilde{\alpha}_{2x}\left[\left.\frac{\partial^2\phi}{\partial^2 x}\right|_c (x-x_c) + \left.\frac{\partial^2\phi}{\partial x\partial y}\right|_c (y-y_c)\right], \tag{I.28}$$

$$\tilde{\phi}_y(x,y) = \left.\frac{\partial\phi}{\partial y}\right|_c + \tilde{\alpha}_{2y}\left[\left.\frac{\partial^2\phi}{\partial^2 y}\right|_c (y-y_c) + \left.\frac{\partial^2\phi}{\partial x\partial y}\right|_c (x-x_c)\right]. \tag{I.29}$$

The parameters $\tilde{\alpha}_i$ are the slope-limiting coefficients for the linear reconstructions, $i \in \{1, 2x, 2y\}$. The reconstruction of the function itself, $\tilde{\phi}_0$, and its first derivatives, $\tilde{\phi}_x$ and $\tilde{\phi}_y$, can be evaluated at the three vertices of each triangular cell. The slope-limiting coefficients are determined such that the value of the reconstructed functions at the vertices do not form local maxima or minima when compared to the cell centre values ($\bar{\phi}$, $\frac{\partial\phi}{\partial x}|_c$, and $\frac{\partial\phi}{\partial y}|_c$) of the linear representations in the neighbouring cells. The neighbour cells considered

when computing the bounds are those that share the vertex, i.e. each vertex has its own unique set of neighbour cells.

The algorithm for computing the $\tilde{\alpha}_i$ slope-limiting coefficients is the same for each of the linear reconstructions in equations (I.27) to (I.29). To simplify the notation, let us work with a generic reconstruction $\tilde{\phi}(x, y)$ limited by coefficient $\tilde{\alpha}_i$ as a stand-in for any one of the linear reconstructions. The corresponding value in the cell centre is denoted $\tilde{\phi}_c = \tilde{\phi}(x_c, y_c)$. For each cell in the mesh, consider each vertex and record the minimum and maximum value of $\tilde{\phi}_c$ in the cells that share the vertex. This gives allowable bounds $\tilde{\phi}_j^{\min}$ and $\tilde{\phi}_j^{\max}$ at vertex j for the selected linear approximation in the given cell. To ensure boundedness, the method imposes that the reconstructed vertex value $\tilde{\phi}_j = \tilde{\phi}(x_j, y_j)$ is bounded by the surrounding cell values, $\tilde{\phi}_j^{\min} \leq \tilde{\phi}_j \leq \tilde{\phi}_j^{\max}$. To find the maximum admissible value of $\tilde{\alpha}_i$ that ensures this, first for each vertex j set

$$\tilde{\alpha}_{ij} = \begin{cases} \min\{1, \frac{\tilde{\phi}_j^{\max} - \tilde{\phi}_c}{\tilde{\phi}_j - \tilde{\phi}_c}\} & \text{if } \tilde{\phi}_j - \tilde{\phi}_c > 0, \\ 1 & \text{if } \tilde{\phi}_j - \tilde{\phi}_c = 0, \\ \min\{1, \frac{\tilde{\phi}_j^{\min} - \tilde{\phi}_c}{\tilde{\phi}_j - \tilde{\phi}_c}\} & \text{if } \tilde{\phi}_j - \tilde{\phi}_c < 0, \end{cases} \tag{I.30}$$

and then compute $\tilde{\alpha}_i = \min_j \tilde{\alpha}_{ij}$. This is performed for each i (each linear reconstruction in equations (I.27) to (I.29) has its own $\tilde{\alpha}_i$) and for each cell in the mesh. The last step is to calculate the final slope limiter coefficients for the second-order derivatives,

$$\alpha_2 = \min\{\tilde{\alpha}_{2x}, \tilde{\alpha}_{2y}\}, \tag{I.31}$$

and, since one can expect higher regularity of the first-order derivatives than the second-order derivatives, take

$$\alpha_1 = \max\{\tilde{\alpha}_1, \alpha_2\}. \tag{I.32}$$

At a smooth extremal point (relative to the mesh density) equation (I.32) will stop any limiting from happening since $\alpha_2 = 1$ here, even if it is likely that $\tilde{\alpha}_1 < 1$. The same hierarchical treatment of the linear reconstructions is performed when working with even higher order basis functions. If a limiting coefficient for the fifth-order derivatives, α_5 is found to be 1.0, then slope-limiting coefficients $\alpha_{1,2,3,4}$ are all 1.0 and the field is considered smooth.

After having computed the slope-limiting coefficients α_1 and α_2 for a given cell, one can project the slope-limited function ϕ_t back to the discontinuous Lagrange function space with the help of the inverse projection $\mathbb{T}^{-1}$. If there were no spurious oscillations in the cell, then the slope-limiting coefficients should all end up as $\tilde{\alpha}_i = 1.0$, and the slope-limiter projection will hence be an identity transform, $\phi^{\lim} = \mathbb{T}^{-1}\,\mathbb{S}\,\mathbb{T}\,\phi = \mathbb{T}^{-1}\,\mathbb{T}\,\phi = \phi$. This ensures that the order of spatial convergence is kept the same as the underlying DG scheme. $\mathbb{S}$ is here the projection from Taylor-basis functions with no limiting to the same basis with limiters computed, the action of $\mathbb{S}$ is to compute and apply $\alpha_{1,2\cdots k} \in [0, 1]$ and $\mathbb{S} = \mathbb{I}$ if and only if $\alpha_{1,2,\cdots,k} = 1.0$.

Slope-limiting algorithm for scalar fields

A summary of the algorithm for slope limiting of a DG scalar field ϕ of any polynomial order by application of a hierarchical procedure, comparing vertex values of linear reconstructions with cell-centre values of neighbouring cells is given below. Everything, except the projections $\mathbb{T}$, are taken from Kuzmin (2010). The algorithm preserves the cell-averaged values of the unlimited scalar field ϕ in the slope-limited scalar field ϕ^{lim}.

1. For each cell in the mesh, project to a Taylor-basis representation of the unlimited scalar field, $\phi_t = \mathbb{T}\phi$, where both the Taylor and Lagrange representations are of the same order k, $\phi \in P_k(K)$.

2. For polynomial orders from 0 up to $k-1$, construct linear reconstructions $\tilde{\phi}_i$ as explained in equations (I.27) to (I.29). The number of reconstructions may be much larger than k since each derivative, $\{{}^{\partial\phi}/_{\partial x}, \cdots, {}^{\partial\phi^{k-1}}/_{\partial x^{k-2}\partial y}, \cdots\}$ requires its own reconstruction.

3. For each reconstruction $\tilde{\phi}_i$, compute the corresponding slope-limiting coefficient $\tilde{\alpha}_i = \min_j \tilde{\alpha}_{ij}$ as explained in equation (I.30) and below. The reconstructed values at the vertices are now bounded by the neighbouring cell-centre values.

4. Compute the slope-limiting coefficient for each derivative order, $\{\alpha_1, \cdots \alpha_k\}$, by taking the minimum of the slope-limiting coefficient for the reconstructions of the same order derivative. Examples: $\alpha_1 = \tilde{\alpha}_1$, $\alpha_2 = \min\{\tilde{\alpha}_{2x}, \tilde{\alpha}_{2y}\}$, and $\alpha_3 = \min\{\tilde{\alpha}_{3xx}, \tilde{\alpha}_{3yy}, \tilde{\alpha}_{3xy}\}$.

5. Increase the slope-limiting coefficients for low-order derivatives if the slope-limiting coefficients for higher-order derivatives are larger. This ensures $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_k$, and hence that smooth maxima are not limited.

6. Project the slope-limited Taylor-basis representation ϕ_t^{lim} back to the discontinuous Lagrange function space, $\phi^{\text{lim}} = \mathbb{T}^{-1}\phi_t^{\text{lim}} = \mathbb{T}^{-1}\mathbb{S}\,\phi_t$, where $\mathbb{S}$ represents the action of steps 2–5 above.

I.3.2 On slope limiting of solenoidal fields

Slope limiting of solenoidal vector fields is more complex than slope limiting of scalar fields due to an increased number of invariants. For scalar fields the only invariant is that the average value in each cell must be unchanged after limiting. For solenoidal vector fields the (i) divergence inside each element and the (ii) flux between neighbouring elements are new invariants in addition to the (iii) average value of each of the velocity component in each cell. Breaking these invariants means that the result is not solenoidal (i), the slope limiting post processor is not a non-local operation (ii), or that momentum is not preserved (iii).

In the following sections we will describe how these invariants can be used to reduce the number of unknowns in the cell-wise limiting problem from 12 to only 4 for a DG2 vector field, $\boldsymbol{u} \in [P_2(K)]^2$, on a triangle. We will then describe an optimisation method which can be used to decide the value of the remaining four unknowns in a way that minimises the tendency to produce local extrema. Removing all local extrema would render the method stable. We do not claim that the below method is optimal, but it does give strong indications that the four remaining unknowns in the cell-wise limiting problem are not sufficient to fully control the local extrema. Creating a single velocity field that is both solenoidal and stable is hence likely not possible—at least not by use of a cell-wise slope-limiting process. As a result of this we propose a new way of slope limiting the velocity field in section I.3.3, which does not require the same field to be both solenoidal and slope limited.

I.3.2.1 A reduced basis for the solenoidal slope-limiting problem

We require the slope limiter to be local to each cell, so we consider only one cell in the following text. Let the cell volume be denoted V_K, and the area of facet i be denoted L_i. Preserving the facet average of the flux between neighbouring cells in the limiting process gives a result which is $\mathrm{H}^{\mathrm{div}}$-compatible in a DG0 sense, which is what is needed for mass conservation. This is the least strict requirement we can put and still maintain the cell-wise locality of the solenoidal limiter. The cell-wise invariants we require to be left unchanged by the slope limiting procedure are then

$$R_i = \frac{1}{L_i}\int_{F_i} \boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}s \quad \forall i \in \{1,2,3\}, \qquad U_j = \frac{1}{V_K}\int_K \boldsymbol{u}_j\,\mathrm{d}x \quad \forall j \in \{1,2\}. \tag{I.33}$$

There are five invariants when applying this to a 2D vector field on a triangle—three average fluxes, denoted R_i, one for each facet, and two cell-average values, denoted U_j, one for each component of the vector field. The sum of facet fluxes for each cell will be unchanged after limiting. This total flux is zero, which is compatible with a solenoidal description of the vector field inside the cell. This property depends on the non-limited vector field being solenoidal, which is a strict requirement of the method.

$$\sum_i R_i = 0 \tag{I.34}$$

The initial 12 degrees of freedom for the non-limited DG2 vector field on a triangle can be reduced since the divergence-free solution will never span the full Lagrange space, but be restricted to the solenoidal subspace which has only 9 degrees of freedom, see e.g. Baker, Jureidini, and Karakashian (1990). This space is spanned by

$$\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}; \quad \begin{pmatrix}y\\0\end{pmatrix}, \begin{pmatrix}0\\x\end{pmatrix}, \begin{pmatrix}x\\-y\end{pmatrix}; \quad \begin{pmatrix}y^2\\0\end{pmatrix}, \begin{pmatrix}0\\x^2\end{pmatrix}, \begin{pmatrix}x^2\\-2xy\end{pmatrix}, \begin{pmatrix}-2xy\\y^2\end{pmatrix}; \tag{I.35}$$

which is here shown grouped into the two zeroth-order, three first-order and four second-order vector valued polynomials.

A projection from the Lagrangian description of $\boldsymbol{u}$ to the solenoidal description $\boldsymbol{u}_s$ can be implemented as $\boldsymbol{u}_s = \mathbb{D}\boldsymbol{u}$ on a cell-by-cell level. A straightforward way of constructing $\mathbb{D}$ is to first define local x and y coordinates with origin in the centre of each cell, and then form the cell-wise rectangular inverse projection $\boldsymbol{u} = \mathbb{D}_I\boldsymbol{u}_s$ by evaluating the solenoidal polynomials from equation (I.35) at the location of the Lagrangian nodes. The least squares pseudo-inverse of this matrix is then $\mathbb{D}$. It is important to note that in our simulations the projection is lossless such that $\boldsymbol{u} = \mathbb{D}_I\mathbb{D}\boldsymbol{u}$. This is due to the $\mathrm{H}^{\mathrm{div}}$ projection, which along with the selected weak form of the momentum and continuity equations causes $\boldsymbol{u}$ to be fully described by the solenoidal subspace, i.e. the non-limited vector field is divergence-free. In general, we must remember that $\mathbb{D}_I \neq \mathbb{D}^{-1}$, so this identity property does not hold for all vector fields.

The velocity vector field can now be described by nine degrees of freedom in each cell, coefficients $s_1 \cdots s_9$. Each of the nine coefficients is multiplied by the corresponding vector valued polynomials in equation (I.35), numbered from left to right. The sum of these products form the complete vector valued polynomial field.

We will further restrict the number of degrees of freedom by making use of the five invariants from equation (I.33). Let the coefficients for the constant and linear polynomials be determined by the five invariants. The four quadratic terms are now the only degrees of freedom left in each cell. Picking four arbitrary numbers, σ_1 to σ_4, for the coefficients s_6 to s_9, one can find corresponding constant and linear weights that make the final vector field keep the selected invariants. In practice this can be done by forming a 9x9 matrix system,

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} & q_{15} & q_{16} & q_{17} & q_{18} & q_{19} \\ q_{21} & q_{22} & q_{23} & q_{24} & q_{25} & q_{26} & q_{27} & q_{28} & q_{29} \\ q_{31} & q_{32} & q_{33} & q_{34} & q_{35} & q_{36} & q_{37} & q_{38} & q_{39} \\ q_{41} & q_{42} & q_{43} & q_{44} & q_{45} & q_{46} & q_{47} & q_{48} & q_{49} \\ q_{51} & q_{52} & q_{53} & q_{54} & q_{55} & q_{56} & q_{57} & q_{58} & q_{59} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \\ s_9 \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \\ R_1 \\ R_2 \\ R_3 \\ \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \end{bmatrix}, \qquad \text{(I.36)}$$

where the first five rows contain appropriate weights q such that the correct integrals from equation (I.33) are computed when the rows are multiplied by s, the solenoidal coefficients. The weights q can easily be computed by quadrature. By solving equation (I.36), the full set of nine solenoidal coefficients, $s_1 \cdots s_9$, can be recovered from the reduced set of four degrees of freedom, $\sigma_1 \cdots \sigma_4$. The original 12 degrees of freedom of the Lagrangian velocity field $\boldsymbol{u}$ can be recovered by use of the projection $\boldsymbol{u} = \mathbb{D}_I\boldsymbol{u}_s$.

We should note that we have no proof that the matrix in equation (I.36) is invertible for all possible cells, but in our extensive testing we have never found

a cell where this was not the case. For other possible matrices where σ do not control the quadratic terms in the solenoidal basis—but let us instead decide that the σ-coefficients are identical to $s_1 \cdots s_4$—then it is easy to find cells where the condition number of the resulting matrix is very high, so the choice of the mapping between the reduced degrees of freedom σ and the solenoidal degrees of freedom s is important.

In this section we have shown how the invariants (i) to (iii) from above leads to a reduction of the slope limiting problem from 12 degrees of freedom in the original DG2 Lagrange space to only four degrees of freedom in the reduced solenoidal space.

I.3.2.2 Optimisation and cost functions

We have used optimisation to determine the four reduced solenoidal degrees of freedom by minimising a cost function in $\mathbb{R}^4$. This optimisation is performed for each cell to find the four coefficients $\sigma_1 \cdots \sigma_4$ that give the most stable solution. This is done with an appropriate choice of cost function that penalises non-smooth local maxima.

An ideal cost function will keep the limited solution very close to the original non-limited solution, while still damping unphysical oscillations. To achieve this we have selected to first calculate a component-wise slope-limited solution $\boldsymbol{u}_{\mathrm{HT}}$ by applying the hierarchical Taylor-polynomial-based slope limiter described in section I.3.1 to each of the velocity components. It should be noted that this makes the result frame dependent. In principle, the scalar limiter could be applied to scalar fields holding the vector field magnitude and direction instead, but that possibility has not been investigated in this work. It is the velocity components themselves that are time integrated, and it is through this time integration that wiggles can grow and destroy the solution, so our focus has been on this instability and not on the introduced frame dependency.

The $\boldsymbol{u}_{\mathrm{HT}}$ field is used as the target in the optimisation since it maintains the correct convergence order. This target field does not keep all the solenoidal invariants and it is hence slightly out of reach of the optimiser where unphysical wiggles are present in the non-limited vector field—the cost function cannot get all the way to zero. The minimum and maximum allowable vector-field component values are computed for a set of points along the cell boundary by considering the average values in the neighbouring cells just like in the hierarchical-Taylor slope limiter. The range of allowable values is extended to include the already limited field $\boldsymbol{u}_{\mathrm{HT}}$ in order to avoid limiting at smooth maxima. The preliminary cost function, without suppression of spurious oscillations, is

$$C_{\mathrm{HT}}(\boldsymbol{x}) = \sum_{i=1}^{2} \left(\frac{u_i - u_{i,\mathrm{HT}}}{\eta_i} \right)^2, \tag{I.37}$$

$$\eta_i = u_{i,\max} - u_{i,\min} \tag{I.38}$$

where the sum is over the vector-field components—hence two contributions in 2D—and η is the width of the allowable range of values.

To penalise deviations from the allowable interval: if the current value u_i is outside the interval an additional cost is incurred. For u_i above $u_{i,\max}$, this is

$$C_\kappa(\boldsymbol{x}) = \sum_{i=1}^{2} \left(\kappa_L \frac{u_i - u_{i,\max}}{\eta_i} \right)^2 + \kappa_C, \tag{I.39}$$

and similar if $u_i < u_{i,\min}$. We have used $\kappa_L = 1.0$ and $\kappa_C = 1.0$ as additional penalties for going outside the allowable range. If the current value u_i is inside the allowable bounds then $\kappa_L = 0$. If both u_1 and u_2 are in the allowable range then $\kappa_C = 0$. The total cost evaluated at a point P_ξ along the perimeter of the cell is

$$C(\boldsymbol{x}_\xi) = C_{\mathrm{HT}}(\boldsymbol{x}_\xi) + C_\kappa(\boldsymbol{x}_\xi) \tag{I.40}$$

The total cost for a cell is the sum of $C(\boldsymbol{x})$ evaluated at a set of points P_ξ along the cell perimeter. We have used the six Lagrangian nodes as the points P_ξ, i.e., the vertices and facet midpoints. The total cost for the cell to be optimised is then calculated as

$$C = \sum_{\xi=1}^{6} \left[C_{\mathrm{HT}}(\boldsymbol{x}_\xi) + C_\kappa(\boldsymbol{x}_\xi) \right]. \tag{I.41}$$

Optimisation-based slope-limiting algorithm for solenoidal vector fields

1. Project the non-limited vector field $\boldsymbol{u}$ to a solenoidal representation in each cell, $\boldsymbol{u}_s = \mathbb{D}\boldsymbol{u}$, where both the solenoidal and Lagrange representations are of the same order k, $\boldsymbol{u} \in [P_k(K)]^d$.

2. Establish the weights q in the matrix from equation (I.36) by an order-appropriate quadrature rule. Use this representation of equation (I.33) to calculate the five invariants from the non-limited vector field in each cell. Both the left- and right-hand sides of equation (I.36) are now known except for the coefficients $\sigma_1 \cdots \sigma_4$.

3. Select a number of points along the perimeter of each cell. We have selected to use the Lagrangian nodes. At each of the points, compute the minimum and maximum allowable values according the neighbouring cells' mean values.

4. At the selected points, compute the component-wise limited $\boldsymbol{u}_{\mathrm{HT}}$ field and use this to extend the range of allowable values to avoid limiting smooth maxima.

5. Use the cost function from equation (I.41) in an optimisation algorithm to find coefficients $[\sigma_1, \sigma_2, \sigma_4, \sigma_4] \in \mathbb{R}^4$ which minimise the cost for a given cell. We have used the Broyden–Fletcher–Goldfarb–Shanno algorithm, BFGS, as implemented in SciPy (Jones, Oliphant, Peterson, et al. 2001).

6. Project the slope-limited solenoidal representation $\boldsymbol{u}_s^{\lim}$ back to the discontinuous Lagrange function space, $\boldsymbol{u}^{\lim} = \mathbb{D}_i \boldsymbol{u}_s^{\lim}$.

I.3.2.3 Optimised velocity fields

The cost function in equation (I.41) is unfortunately not able to steer even a perfect optimisation routine towards a solution that is completely free from spurious local maxima in all cells. There are three times more contributions to the cost function than the degrees of freedom $\sigma_1 \cdots \sigma_4$ in each cell—and that is when the cost function is looking for overshoots only in the Lagrange nodes, there might still be local maxima between the Lagrange nodes.

Detailed studies of the cost functions for some selected problematic cells show that there are cells where there does not exist any point in the four dimensional coefficient space $\sigma_1 \cdots \sigma_4$ where the cost is below $\kappa_C = 1.0$. Using a brute force optimisation algorithm that explores the entire solution space would not help for those cells. In conclusion, it is likely not possible—by cell-wise slope limiting—to reconstruct a velocity field that is both solenoidal and free from local maxima based on the presented reduced basis and the chosen criteria to detect local overshoots.

I.3.3 A split solenoidal slope-limiting algorithm

Motivated by the above findings we introduce separate limiters for the *convected* velocity $\boldsymbol{u}$ and the *convecting* velocity $\boldsymbol{w}$. Looking at the convected and convecting velocity fields as separate, but related, is inspired by the common practice of using an explicit convecting velocity to linearise the Navier–Stokes equations—which is also what we have done in equation (I.15). A similar splitting is done in cell-centred finite volume schemes where the convecting velocity field is interpolated to the facets while the unknown convected velocity field consists of cell averages. The convect*ing* velocity can be partially slope limited in a way that does not compromise on the solenoidal properties, or left entirely unlimited. The convect*ed* velocity field must be slope limited to avoid instabilities. The overall algorithm is shown in figure I.2.

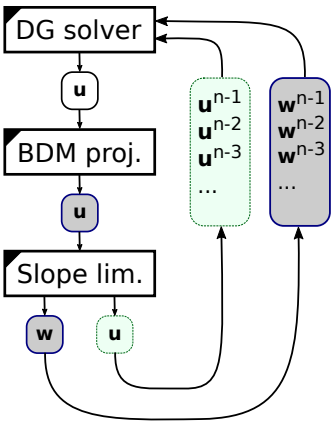


Figure I.2: The slope-limiting algorithm splits the velocity field into two separate time histories. Shaded velocities are solenoidal when the box boundary is a continuous line (grey background) and free from local maxima when the line is dotted (light green background).

In the following results section we have applied three different slope limiting procedures for the velocity field. The first is the naive unlimited method, the second applies the hierarchical Taylor-polynomial-based limiter to each velocity component of the convected velocity, and applies no limiting to the convecting velocity. The third method applies slightly different versions of the solenoidal slope limiter from section I.3.2.2 to the two velocity fields. The convecting velocity field is limited without regards for the resulting cost function value in each cell; hence, for some cells the suppression of local maxima will not be successful. The convected velocity field is treated similarly, but here the cell velocity fields are replaced with the component-wise limited fields from the hierarchical Taylor-polynomial-based limiter, $\boldsymbol{u}_{\mathrm{HT}}$, in the cells where the local maxima suppression is unsuccessful.

I.4 Results

The methods described above have been implemented in Ocellaris (Landet 2019), a two-phase solver framework which is built on top of FEniCS (Logg, Mardal, and Wells 2012) and implemented in a mix of Python and C++. The optimiser employed is the BFGS algorithm as implemented in SciPy (Jones, Oliphant, Peterson, et al. 2001), but for efficiency reasons we have written the optimisation algorithm along with the cost function in C++ to decrease the running time of the solver. The hierarchical Taylor-based slope limiter for scalar fields has been implemented in C++ and does not contribute significantly to the running time of the solver. The rest of the Ocellaris solver is implemented in Python and depends on the C++ code-generation facilities in FEniCS to utilise the computational resources optimally. The source code, input files, and scripts to reproduce the figures shown below can be found in Landet 2017.

In the following text we will compare three numerical algorithms, (i) a naive DG implementation without any slope limiters, (ii) a simple velocity slope limiting implementation using the scalar Taylor-based limiter on each of the convected velocity components and no limiting of the convecting velocity (referred to as 'Hierarchical Taylor'), and (iii) the more involved cell-wise optimisation described above where both the convected and convecting velocity fields are optimised with slightly different criteria (referred to as 'Solenoidal').

To properly test the presented limiters it is important to ensure that artificially high viscosity is not a contributing factor to the method's stability. For the air/water free-surface test cases we have applied $\nu = 1.0 \times 10^{-6}\,\mathrm{m^2\,s^{-1}}$ for both phases, which is artificially low for the air phase. With this choice of viscosity, the non-linear convective instability impacts the results early in the simulations. This is also true for, e.g., $\nu = 1.0 \times 10^{-4}\,\mathrm{m^2\,s^{-1}}$, but not for very high kinematic viscosities, when ν approaches 1.0. Note that the dynamic viscosity μ—the parameter that goes into the weak form in equation (I.20)—is not the same in the two phases, it has the same factor 1000 jump as the density.

I.4.1 Taylor-Green vortex

The first test case is a study of the effect of slope limiting on the spatial convergence of the numerical scheme by considering a Taylor-Green vortex where the density is constant and the solution $\boldsymbol{u} = [u, v], p$ is given by,

$$\begin{aligned} u &= -\sin(\pi y)\cos(\pi x)\exp(-2\pi^2\nu t) \\ v &= \sin(\pi x)\cos(\pi y)\exp(-2\pi^2\nu t) \\ p &= -1/4\,\rho(\cos 2\pi x + \cos 2\pi y)\exp(-4\pi^2\nu t) \end{aligned} \tag{I.42}$$

for $t \in [0, 1]$ on a domain $\Omega = \{(x, y) \in [0, 2] \times [0, 2]\}$. This test case is often solved on a periodic domain, but we use Dirichlet boundary conditions for $\boldsymbol{u}$ to test the effect of the boundary. Initial conditions are given both at $t = 0$ and $t = -\Delta t$ to be able to use second-order time stepping from the start. A constant time step of $\Delta t = 0.01$ is used and the kinematic viscosity is $\nu = 0.005$.

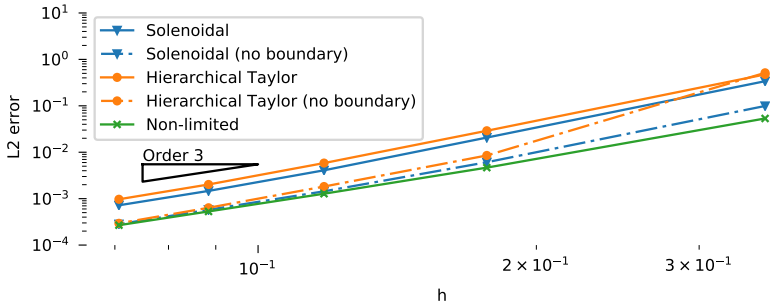


Figure I.3: Spatial convergence rate on the Taylor-Green vortex test case. The rate of convergence is as expected, but missing neighbour-cell information near boundaries in the limiters gives a larger constant in the L^2 error plot.

A slope limited solution should be identical to the non-limited solution for this test case—there are no non-smooth maxima in the true solution. Still, exact equivalence is not obtained since the DG FEM vector field is only a numerical approximation. The main discrepancy is in the handling of boundary conditions. Vertices on the boundary will be missing neighbour cells and may hence be unnecessarily slope limited. Figure I.3 shows that the results converge towards the non-limited solution when slope limiting is avoided in the boundary facing cells. When boundary cells are included in the limiter, the expected third-order L^2 convergence rate is still obtained, but the constant is larger.

I.4.2 Dam break

The first two-phase flow example is a classic dam-break-in-a-box simulation as illustrated in figure I.4 based on the experiments by Martin and Moyce (1952). This is one of the most commonly used test cases in the high-Reynolds-number low-surface-tension regime. This is the regime most interesting for studying marine and offshore structures, which is the ultimate goal of the method.

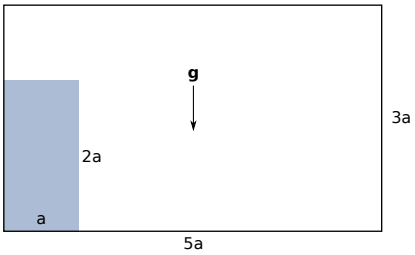


Figure I.4: Dam-break simulation geometry.

The 2D rectangular water column is $1a$ wide and $2a$ high and starts at rest in the lower left corner of a box filled with air which is $5a$ wide and $3a$ high. The size of the column is the same as in the experiments, $a = 2.25\,\text{in} = 0.057\,15\,\text{m}$. The two phases are water and air with densities $1000\,\text{kg}\,\text{m}^{-3}$ and $1.0\,\text{kg}\,\text{m}^{-3}$ respectively. This corresponds to tables 2 and 6 in Martin and Moyce (1952). The acceleration of gravity is $g = 9.81\,\text{m}\,\text{s}^{-2}$ in the negative y-direction and the kinematic viscosity is $\nu = 1.0 \times 10^{-6}\,\text{m}^2\,\text{s}^{-1}$ for both phases.

The total kinematic energy, $E_k = \int_\Omega {}^1\!/_2\,\rho\,\boldsymbol{u}\cdot\boldsymbol{u}\,\mathrm{d}\boldsymbol{x}$, is shown in figure I.5 as a function of time for the slope-limited and the non-limited methods. From the start, with both fluids at rest, the kinetic energy increases as the water mass starts to flow down and towards the right wall. There is a slight reduction as the water hits this wall at approximately $t = 0.19\,\text{s}$. The Gibbs oscillations start to dominate the non-limited solution after a short time and from $t = 0.04\,\text{s}$ the solution is non-physical and completely dominated by numerical errors. In the rest of this paper we will remove the non-limited method from the results as it leads to non-physical solutions in all cases.

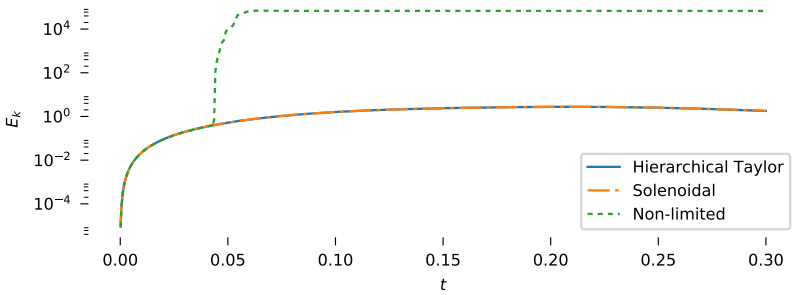
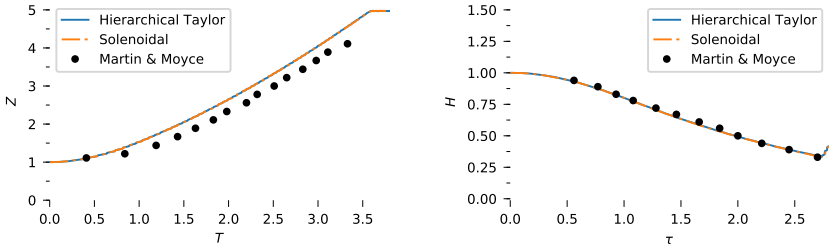


Figure I.5: Kinetic energy as a function of time for three slope-limiting strategies.

The results of the two slope-limited methods are compared to the experimental results in figure I.6. The experimental data points are the ensemble averages of the results reported for the same geometry by Martin and Moyce (1952). We have employed free-slip boundary conditions, and hence the surge front moves slightly faster than in the experimental results, but the qualitative behaviour is



(a) Surge front position (b) Height of water column

Figure I.6: Comparison of the numerical results with experimental results by Martin and Moyce (1952). The vertical axis scaling is $Z = z/a$ and $H = \eta/2a$. Measured from the lower left corner, z is the surge front position and η is the height of the water column. The horizontal axis scaling is $T = t\sqrt{2g/a}$ and $\tau = t\sqrt{g/a}$.

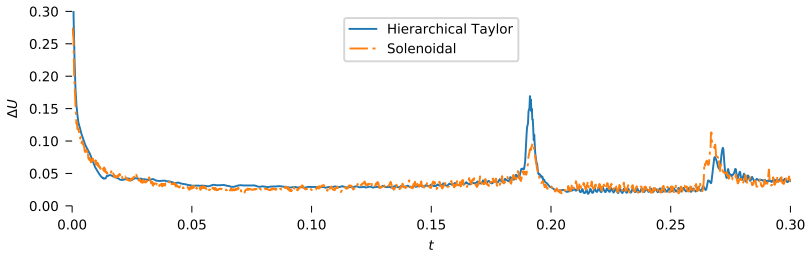


Figure I.7: Dam break; comparison of the two velocity slope limiters in terms of, ΔU, the scaled L^2 distance between convected and convecting velocity fields.

correct. The maximum height of the water column matches very well until the water hits the domain boundary and creates a jet shooting up along the right wall at $t \approx 0.19\,\mathrm{s}$ which corresponds to $T = t\sqrt{2g/a} \approx 3.5$ and $\tau = t\sqrt{g/a} \approx 2.8$. The right wall was placed much further away in the experimental setup.

As can be seen in figures I.5 and I.6, the two investigated limiters perform very similarly on the dam break test case. The 'Solenoidal' method requires significantly more computation per time step than the component-wise Taylor-based limiter and is more complex to implement. One reason why one could still consider using this method is that the optimisation may keep the difference between the convected and the convecting velocity fields smaller and hence closer to the true solution. To test how much this optimisation contributes we have calculated the difference between the velocity fields, $\Delta U = \|\boldsymbol{u} - \boldsymbol{w}\| \,/\, \|\boldsymbol{u}\|$, for all time steps and the results can be seen in figure I.7.

The difference is large in the beginning since both $\boldsymbol{u}$ and $\boldsymbol{w}$ are close to zero at this time. After the initial phase the difference is low for both limiters. We can see that right before the water impacts the wall at approximately $t = 0.19\,\mathrm{s}$, the slope limiters start to perform differently for some time until the impact

is over. The reason is that the 'Hierarchical Taylor' slope limiter introduces a very high divergence in the convected velocity field in the tank corner. The optimisation based limiter performs as expected and the difference between the two fields is lower, but not entirely removed.

Later in the time series there is a new event when the water is shooting up in a thin jet along the right wall at approximately $t = 0.27\,\mathrm{s}$, corresponding to slightly after the illustration in figure I.1c. In figure I.7 one can see that the optimised slope limiter is introducing slightly higher ΔU differences than the Taylor-based slope limiter, quite contrary to the intention. This is actually a pathological case where the difficulties inherent in trying to conserve both the cell-averaged values and the facet-averaged fluxes become quite clearly visible.

To explain the situation, let the width of the jet be denoted δ, and let us assume that the velocity in the air outside the jet is much lower than the velocity of the water inside the jet, both purely vertical. The width of the jet is significantly smaller than the width of the cells close to the vertical wall where the jet is shooting up. Since not all facets in the triangular cells can be either orthogonal or parallel to the flow, the part of each cell's area that is covered by the jet is proportional to δ^2, which in turns requires the slope-limited magnitude of the jet to be proportional to δ^{-2} in order to preserve the average vertical velocity in each cell. The problem is that the length of the parts of the facets covered by the jet are linearly related to the jet width, δ, so smoothing the sharp gradient by widening the jet will change the facet fluxes, which is not allowable in the optimisation-based solenoidal slope limiter. The optimiser will have to come up with a solution that is smooth, and also does not change the facet fluxes, not an easy task. The end result is that the difference between the optimised solution and the component-wise limited solution is larger than the difference between the non-limited solution and the component-wise solution, as can be seen in figure I.7.

I.4.3 Tank filling

To provide a more challenging test of the method's stability with a large and complex free surface we have performed a set of tank-filling simulations inspired by Guermond, Luna, and Thompson (2017), but unlike in their work we have used physical properties closer to normal water and air, and the results are hence more energetic. The physical properties are the same as in the dam break test case.

The geometry of the tank can be seen along with the resulting density field in figure I.8. The domain is the unit square and the inlet is located between 0.5 m and 0.625 m above the floor on the left wall, 1/8 m wide. The outlet is located in the roof between 0.5 m and 0.625 m from the left wall and is also 1/8 m wide. The inlet and outlet velocities are prescribed as $2\,\mathrm{m\,s^{-1}}$, constant for all time. At t=0 the tank is completely filled with air, the velocity inside is zero, and the pressure is hydrostatic.

The results are computed on a regular mesh with triangular cells and no mesh grading. There are eight cells across the inlet and 64x64x2 cells in total.

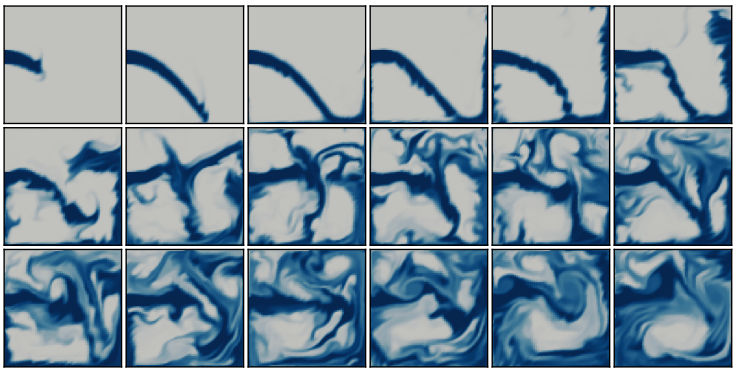
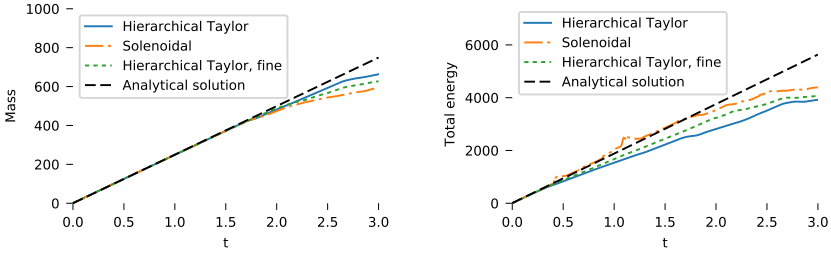


Figure I.8: Tank-filling results. The first row shows the density field at $t = 0.167$, 0.333, 0.500, 0.667, 0.833, and 1.000, the second and third row have the same temporal spacing, hence the lower rightmost image shows the simulation end point, $t = 3$ s. The finest resolution is shown, using the 'Hierarchical Taylor' slope limiter.

The domain is divided into squares which are further subdivided into triangles. The time step is $\Delta t = 0.0001$ s. The solution has also been calculated on a finer mesh with 16 elements across the inlet, 128x128x2 cells in total. The time step was then $\Delta t = 5.0 \times 10^{-5}$ s and only the 'Hierarchical Taylor' slope limiter was tested. The results from all three simulations qualitatively exhibit the same physics and look similar. The density field from the simulations on the fine mesh is shown in figure I.8 at regular intervals up to the maximum simulation time of 3 s.



(a) Total mass (b) Total energy

Figure I.9: Tank filling: comparison of the two velocity slope limiters in terms of conservation of mass and energy.

The performance of the slope limiters is compared in terms of stability and conservation. We have calculated the total mass and energy inside the tank and the results can be seen in figures I.9a and I.9b. The analytical solutions in the figures are computed under the assumption that only air exits through the outlet, which is not true in the second half of the simulation. When looking

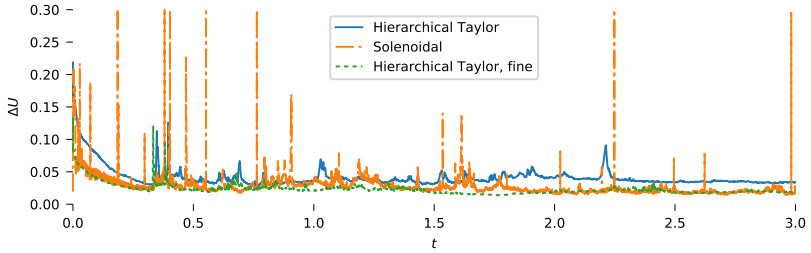


Figure I.10: Tank filling: comparison of the two velocity slope limiters in terms of, ΔU, the scaled L^2 distance between convected and convecting velocity fields.

at conservation of mass in the first part of the simulations, only very minor differences can be seen between the methods; both methods preserve mass very well. This is exactly as expected since mass conservation is the key invariant in both slope limiters—the convecting velocity field is always solenoidal.

The loss of mass and energy through the outlet—starting at approximately 1.7 seconds—can also be seen in the time history of the total energy in figure I.9b. Up to that point the optimisation-based solenoidal slope limiter can be seen to preserve total energy better than the component-wise hierarchical Taylor-based limiter. Refining the mesh improves the results from the hierarchical Taylor-based limiter, but the optimised solenoidal limiter still outperforms it, even if it is running on a much coarser mesh.

Both methods are stable for the duration of the simulation, but some spurious increases in total energy can be seen in the 'Solenoidal' method. The 'Hierarchical Taylor' method has no noticeable spurious increases, at the expense of slowly loosing energy. The reason for the spurious increase in energy in the 'Solenoidal' method around 1.2 s in figure I.9b may be the fact that local maxima are allowed to occur between the Lagrange polynomial nodes.

The difference between the convected and the convecting velocity, ΔU, has been calculated in the same manner as in the dam-break test case and the results can be seen in figure I.10. Both methods preserve a low difference between the velocity fields for most of the time steps, but both have instances when the difference grows larger. This is particularly noticeable in the 'Solenoidal' method which has some large spikes. In general the optimised method maintains a lower difference, on par with the refined hierarchical Taylor-based method. Both methods are shown to recover from temporary spikes and return to a low difference, which is the correct solution.

I.5 Discussion

We have shown how convective instabilities in higher-order two-phase flow simulations can be stabilised without sacrificing mass conservation. Our efforts have been focused on the use of slope limiters to achieve this stabilisation. Using optimisation on a solenoidal reduced basis to create one single velocity field

that is both stable and solenoidal was investigated. The results show strong indications that this may in fact not be possible. To overcome this problem, two methods that treats the convected velocity field differently from the convecting velocity field were proposed. These methods combine exact mass and momentum conservation with convective stability for two-phase simulations containing large density jumps inside the computational domain.

The suggested methods have been tested to see if there were any negative impact on the solution of smooth problems. Optimal convergence on a smooth Taylor-Green vortex problem was shown. Further, a two-dimensional dam-break simulation with a factor-1000 sharp density jump in the computational domain was tested. Such simulations are not possible to handle without some form of convective stabilisation of the higher-order solution. Both slope limiting strategies handled this problem well. Finally, we have studied mass and energy conservation in a very energetic flow—a tank-filling simulation. A summary of our results is that slope limiting is a possible way to stabilise a mass-conserving discontinuous Galerkin method for the incompressible Navier–Stokes equation with large density jumps.

There are other methods than slope limiting for stabilising high-order convective instabilities in discontinuous Galerkin methods. Instead of using an explicit post processing operator, one can add specifically tailored implicit damping to the cells near the free surface with a method such as the entropy-viscosity method, (Zingan et al. 2013). Another method that has been used is to selectively reduce the polynomial approximation order in cells near the free surface, a special use of p-adaptivity (Robert J. Labeur, private communication, 2017). The advantages of the explicit slope-limiting method is the decoupling of the convective stabilisation from the definition of the weak form and the assembly of the equation system.

The component-by-component velocity-limiting method has a large advantage over other available methods in the relative ease of implementation. Some type of scalar slope limiter is likely to be already present in a DG FEM code. Just as for the solenoidal limiter, no extra method-dependent viscosity parameters need to be derived, and no selective p-adaptivity is needed in the software. There is no need to tune parameters in the methods to achieve stability. Extension to higher-order is also straightforward.

The run-time penalty of the component-by-component velocity-limiting method is small. Even on realistic 3D simulations with MPI-parallelised Krylov-solvers (upcoming paper), instead of the relatively slow 2D direct coupled equation solver used here, we still observe that the run-time penalty is negligible, around 0.3 % of the wall clock time, though this relative contribution may increase somewhat as the overall solver is further optimised. Based on the results of this paper, we have not included the solenoidal limiter in the parallelised version of the solver, so we do not have realistic performance numbers for this. With a single-CPU direct solver, as used in this paper, the Solenoidal slope limiter is far from dominating the running time, but it is obviously much more expensive than the component-by-component velocity-limiting method.

Both slope-limiter methods require access to degrees of freedom in neighbour

cells, even if the calculations only change degrees of freedom in one cell at a time. This is efficiently handled through mesh ghosting when running in parallel on a computational cluster. The DG method itself requires access to degrees of freedom in cells sharing a facet to compute fluxes, so the only change is to require node-neighbour ghosting instead of only facet-neighbour ghosting. This will have some impact on performance due to larger message sizes every time the MPI ranks are synchronised.

I.6 Conclusion

An algorithm using cell-wise optimisation in an attempt to construct a slope limiting method for solenoidal vector fields is presented. The goal of the algorithm is to keep the divergence free property, while removing unphysical wiggles. We show strong indications that obtaining both machine-zero divergence and convective stability is not possible in a slope limiting post-processing filter with a single velocity field. The solenoidal limiter is then reformulated in a framework where the convect*ing* and the convect*ed* velocity fields are treated separately. In this framework, any existing scalar slope limiter can be used on each of the velocity components, dramatically simplifying the implementation. The solenoidal slope limiter is compared with an existing hierarchical Taylor-polynomial-based scalar slope limiter, and the results indicate that component-wise limiting gives equally good results. Component-wise limiting can be recommended as the simplest and least intrusive way of stabilising a single-phase DG FEM solver for use in a multi-phase flow setting where the density field has a sharp discontinuity and the convecting velocity field should remain exactly solenoidal.

Acknowledgements

The authors are thankful to Miroslav Kuchta for proofreading and for valuable discussion related to this work. We would also like to thank the reviewers, especially the descriptions of the slope-limiting methods have been greatly improved by their comments.

References

Arnold, D. N. (1982). “An interior penalty finite element method with discontinuous elements”. *SIAM journal on numerical analysis* 19.4, pp. 742–760.

Baker, G. A., Jureidini, W. N., and Karakashian, O. A. (Dec. 1990). “Piecewise Solenoidal Vector Fields and the Stokes Problem”. *SIAM Journal on Numerical Analysis* 27.6, pp. 1466–1485.

Brezzi, F. and Fortin, M. (1991). *Mixed and Hybrid Finite Element Methods*. Springer Series in Computational Mathematics 15. Springer-Verlag, New York.

Cockburn, B., Kanschat, G., and Schötzau, D. (2004). "The local discontinuous Galerkin method for the Oseen equations". *Mathematics of Computation* 73.246, pp. 569–594.

— (2005). "A locally conservative LDG method for the incompressible Navier-Stokes equations". *Mathematics of Computation* 74.251, pp. 1067–1096.

— (2007). "A Note on Discontinuous Galerkin Divergence-free Solutions of the Navier–Stokes Equations". *Journal of Scientific Computing* 31.1-2, pp. 61–73.

Cockburn, B. and Shu, C.-W. (Apr. 1998). "The Runge–Kutta Discontinuous Galerkin Method for Conservation Laws V". *Journal of Computational Physics* 141.2, pp. 199–224.

— (2001). "Runge-Kutta discontinuous Galerkin methods for convection-dominated problems". *Journal of Scientific Computing* 16.3, pp. 173–261.

Epshteyn, Y. and Rivière, B. (2007). "Estimation of penalty parameters for symmetric interior penalty Galerkin methods". *Journal of Computational and Applied Mathematics* 206.2, pp. 843–872.

Guermond, J.-L., Luna, M. Q. de, and Thompson, T. (Sept. 2017). "An conservative anti-diffusion technique for the level set method". *Journal of Computational and Applied Mathematics* 321, pp. 448–468.

Guermond, J.-L. and Quartapelle, L. (Nov. 2000). "A Projection FEM for Variable Density Incompressible Flows". *Journal of Computational Physics* 165.1, pp. 167–188.

Guermond, J.-L. and Salgado, A. (May 2009). "A splitting method for incompressible flows with variable density based on a pressure Poisson equation". *Journal of Computational Physics* 228.8, pp. 2834–2846.

Hirt, C. W. and Nichols, B. D. (1981). "Volume of fluid (VOF) method for the dynamics of free boundaries". *Journal of computational physics* 39.1, pp. 201–225.

Hundsdorfer, W., Ruuth, S. J., and Spiteri, R. J. (Jan. 2003). "Monotonicity-Preserving Linear Multistep Methods". *SIAM Journal on Numerical Analysis* 41.2, pp. 605–623.

Jones, E., Oliphant, T., Peterson, P., et al. (2001). *SciPy: Open source scientific tools for Python*.

Kirby, R. C. et al. (2012). "Common and unusual finite elements". In: *Automated Solution of Differential Equations by the Finite Element Method*. Lecture Notes in Computational Science and Engineering. Springer, Berlin, Heidelberg, pp. 95–119.

Kuzmin, D. (Apr. 2010). "A vertex-based hierarchical slope limiter for p-adaptive discontinuous Galerkin methods". *Journal of Computational and Applied Mathematics*. Finite Element Methods in Engineering and Science (FEMTEC 2009) 233.12, pp. 3077–3085.

— (Mar. 2013). "Slope limiting for discontinuous Galerkin approximations with a possibly non-orthogonal Taylor basis". *International Journal for Numerical Methods in Fluids* 71.9, pp. 1178–1190.

Landet, T. (July 2017). *Ocellaris DG FEM software and input files to reproduce results*. Zenodo: 10.5281/zenodo.845352.

Landet, T. (2019). "Ocellaris: a DG FEM solver for free-surface flows". *Journal of Open Source Software* 4.35, p. 1239.

Leonard, B. P. (1988). "Simple high-accuracy resolution program for convective modelling of discontinuities". *International journal for numerical methods in fluids* 8.10, pp. 1291–1318.

Liu, C. and Walkington, N. J. (Jan. 2007). "Convergence of Numerical Approximations of the Incompressible Navier–Stokes Equations with Variable Density and Viscosity". *SIAM Journal on Numerical Analysis* 45.3, pp. 1287–1304.

Logg, A., Mardal, K.-A., and Wells, G. (Feb. 2012). *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Springer Science & Business Media.

Martin, J. C. and Moyce, W. J. (Mar. 1952). "Part IV. An Experimental Study of the Collapse of Liquid Columns on a Rigid Horizontal Plane". *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 244.882, pp. 312–324.

Michoski, C. et al. (Jan. 2016). "A Comparison of Artificial Viscosity, Limiters, and Filters, for High Order Discontinuous Galerkin Solutions in Nonlinear Settings". en. *Journal of Scientific Computing* 66.1, pp. 406–434.

Muzaferija, S. et al. (1999). "A Two-Fluid Navier-Stokes Solver to Simulate Water Entry". In: *Twenty-Second Symposium on Naval Hydrodynamics*. Washington, DC: The National Academies Press, pp. 638–651.

Nédélec, J.-C. (Jan. 1986). "A new family of mixed finite elements in $\mathbb{R}^3$". *Numerische Mathematik* 50.1, pp. 57–81.

Nitsche, J. A. (July 1971). "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* 36.1, pp. 9–15.

Olsson, E. and Kreiss, G. (Nov. 2005). "A conservative level set method for two phase flow". *Journal of Computational Physics* 210.1, pp. 225–246.

Osher, S. and Sethian, J. A. (1988). "Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations". *Journal of computational physics* 79.1, pp. 12–49.

Peskin, C. S. (Jan. 2002). "The immersed boundary method". *Acta Numerica* 11, pp. 479–517.

Pyo, J.-H. and Shen, J. (Jan. 2007). "Gauge–Uzawa methods for incompressible flows with variable density". *Journal of Computational Physics* 221.1, pp. 181–197.

Shahbazi, K., Fischer, P. F., and Ethier, C. R. (2007). "A high-order discontinuous Galerkin method for the unsteady incompressible Navier-Stokes equations". *Journal of computational physics* 222.1, pp. 391–407.

Sussman, M., Smereka, P., and Osher, S. (Sept. 1994). "A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow". *Journal of Computational Physics* 114.1, pp. 146–159.

Sweby, P. K. (1984). "High resolution schemes using flux limiters for hyperbolic conservation laws". *SIAM Journal on Numerical Analysis* 21.5, pp. 995–1011.

Touré, M. K., Fahsi, A., and Soulaïmani, A. (Jan. 2016). "Stabilised finite-element methods for solving the level set equation with mass conservation". *International Journal of Computational Fluid Dynamics* 30.1, pp. 38–55.

Ubbink, O. (1997). "Numerical prediction of two fluid systems with sharp interfaces". PhD thesis. Imperial College, University of London.

Unverdi, S. O. and Tryggvason, G. (May 1992). "A front-tracking method for viscous, incompressible, multi-fluid flows". *Journal of Computational Physics* 100.1, pp. 25–37.

Zingan, V. et al. (Jan. 2013). "Implementation of the entropy viscosity method with the discontinuous Galerkin method". *Computer Methods in Applied Mechanics and Engineering* 253, pp. 479–490.

II

Paper II

On exactly incompressible DG FEM pressure-splitting schemes for the Navier–Stokes equation

Tormod Landet, Mikael Mortensen

Abstract

We compare three existing iterative pressure-correction schemes for solving the Navier–Stokes equations with a focus on obtaining exactly divergence-free solutions with a higher-order discontinuous Galerkin discretisation. The investigated schemes are the incremental pressure-correction scheme on the standard differential form (IPCS-D), the same scheme on algebraic form (IPCS-A), and the semi-implicit method for pressure-linked equations (SIMPLE).

We show algebraically and through numerical examples that the IPCS-A and SIMPLE schemes can be exactly mass conserving due to the algebraic pressure correction, while the IPCS-D scheme can never give exactly divergence free results due to the stabilisation terms required in the pressure-Poisson equation. By exact mass conservation we mean that the velocity field is pointwise divergence free inside each mesh cell and that the normal velocity is single valued across internal mesh facets. The SIMPLE scheme requires a significantly higher number of pressure-correction iterations to obtain converged results than the IPCS-A scheme, so for efficient and mass conserving DG FEM simulations the IPCS-A method is the best option among the three methods evaluated.

II.1 Introduction

The incompressible Navier–Stokes equations are challenging to solve efficiently. Fast, stable, and parallel solution procedures are still an active research field more than fifty years after the first pressure-correction solvers were introduced by Chorin (1968) and Temam (1969). Our focus in this work is on how to obtain machine-precision zero divergence in the resulting velocity field when applying iterative solvers. This is possible since the selected DG FEM discretisation has this property when the system is solved by a direct solver. We approach the topic by comparing three well-known pressure-correction methods. Our contribution is the comparison of the methods in terms of not only speed and accuracy, but also the resulting divergence. This work is meant to help select a pressure-correction solver among the most popular methods for uses where the divergence-free properties of the original system is wanted also in the iterative pressure-correction method.

The equation system to be solved is the linearised incompressible Navier–Stokes equations, where we have removed the non-linearity simply by introducing an explicitly extrapolated convecting velocity. This is done since the non-linearity is not the topic of this investigation, and this is also a common way of dealing with the convective term in the investigated methods. Denoting the unknown velocity vector as $\boldsymbol{u}$, the explicit convecting velocity as $\boldsymbol{w}$, the pressure as p, the fluid density as ρ and the fluid dynamic viscosity as μ, the equations can be written

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{w}\cdot\nabla)\boldsymbol{u}\right) = \nabla\cdot\mu\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right) - \nabla p, \tag{II.1}$$

$$\nabla\cdot\boldsymbol{u} = 0,$$

which, after discretisation, leads to a sparse discrete block matrix problem,

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{u}\\ \mathbf{p}\end{bmatrix} = \begin{bmatrix}\mathbf{d}\\ \mathbf{e}\end{bmatrix}, \tag{II.2}$$

where the sub-matrix $\mathbf{A}$ is a discrete version of the bilinear operator $\mathcal{A}$,

$$\mathcal{A}\bullet \equiv \frac{\rho}{\Delta t}\gamma_1\bullet + \rho(\boldsymbol{w}\cdot\nabla)\bullet - \nabla\cdot\mu\left(\nabla\bullet + (\nabla\bullet)^T\right), \tag{II.3}$$

which has been assembled by use of the discontinuous Galerkin finite element (DG FEM) scheme described in section II.2. Similar holds for the matrices $\mathbf{B}$ and $\mathbf{C}$, the discrete versions of the pressure-gradient and velocity-divergence operators. The unknowns $\mathbf{u}$ and $\mathbf{p}$ (bold, non-italic) are now arrays of unknowns containing the degrees of freedom describing the velocity and pressure fields. The vectors $\mathbf{d}$ and $\mathbf{e}$ contain the assembled linear operators from the momentum and continuity equations respectively. Note that due to the application of boundary conditions, the vector $\mathbf{e}$ is not necessarily zero.

There are many methods for solving the sparse saddle-point system shown in equation (II.2). With appropriate boundary and initial conditions the equations are well-formed and can be solved directly on coupled form by an LU-decomposition method. Fast parallel sparse LU solvers exist, such as MUMPS,

which uses multifrontal LU factorization (Amestoy, Duff, et al. 2001; Amestoy, Guermouche, et al. 2006), and SuperLU_DIST, which uses supernodal LU factorization (X. Li et al. 1999; X. S. Li 2005; X. S. Li and Demmel 2003). Unfortunately, such direct solution methods only scale up to a relatively small number of parallel processors, but they can solve this saddle-point linear equation system without any special preconditioning.

Unlike direct solvers, iterative Krylov-subspace methods can scale to thousands of processors, but they require preconditioning for efficient and stable solution of saddle-point problems (Fletcher 1976; Hestenes and Stiefel 1952; Saad and Schultz 1986; Vorst 1992). Pressure-correction methods are a popular family of methods to deal with the saddle-point problem in the Navier–Stokes equations by splitting the coupled equations into individual momentum-guess and pressure-correction steps, which can then be solved efficiently by sparse iterative Krylov solvers. The earliest pressure-correction schemes are those by Chorin (1968) and Temam (1969), but the splitting error introduced in these can only be reduced by using smaller time steps. Iterative pressure correction methods, such as the incremental pressure-correction scheme, IPCS (Goda 1979; Kan 1986), control the splitting error separately by optionally performing multiple pressure corrections for each time step.

An overview of the literature related to pressure-correction methods can be found in, e.g., Badia and Codina (2007) and J. L. Guermond, Minev, and Shen (2006). The two main methods for implementing the equation splitting are the differential splitting method, where the strong form in equation (II.1) is split before discretisation, and the algebraic splitting method, which operates on the discretised form in equation (II.2). One important difference between algebraic and differential pressure-correction methods is the treatment of boundaries. Differential methods—also known as continuous methods—require application of boundary conditions when solving the pressure-correction equation. Algebraic methods do not require any extra boundary conditions, but that does not mean that the error in the pressure due to the splitting is lower. Both types of splitting methods introduce splitting errors that typically show up most clearly near the boundaries. J. L. Guermond, Minev, and Shen (2006) show that algebraic versions of Chorin/Temam type splitting schemes enforce a Neumann boundary condition (weakly) on the pressure where the velocity has a Dirichlet boundary condition, just like the differential versions do explicitly when constructing the weak form of the pressure-Poisson equation and applying boundary conditions derived from the normal component of the velocity boundary condition.

Using the normal component of the viscous wall boundary condition, $\boldsymbol{u}\cdot\boldsymbol{n} = 0$, to construct the boundary condition for the pressure is compatible with the continuity equation. This enables the exact incompressibility that we look for in this comparison paper. It is possible to include the tangential velocity boundary condition and obtain a Dirichlet boundary condition for the pressure as well, see e.g. Orszag, Israeli, and Deville (1986). Using a Neumann boundary condition for the pressure-Poisson equation is by far the most popular choice, and Sani et al. (2006) show that when the Poisson equation for the pressure is solved along with the momentum equation in a coupled manner, the results are exactly

the same as when the regular continuity equation is used instead—as long as Neumann boundary conditions are applied for the pressure based on the normal component of the velocity Dirichlet boundary condition. Since the equations are not solved simultaneously, but through a time-splitting algorithm, there will be errors that manifest most strongly near the boundaries. Karniadakis, Israeli, and Orszag (1991) show that the thickness of the layer of numerical pressure-splitting errors close to the boundary is proportional to $\sqrt{\nu\gamma_1\Delta t}$. They also show that the order of the time-differencing error in the velocity is at most one order greater than the error in the boundary conditions, so a second order time differencing scheme requires at least a first order treatment of the boundary conditions of the pressure. It is this combination that is used in this paper, a second order backwards differencing formulation, BDF2.

Solving the Navier–Stokes equations will in general require a choice of function spaces for the velocity and pressure that satisfy an inf–sup stability criterion—often called the Ladyzhenskaya–Babuška–Brezzi (LBB) condition for conforming finite elements. See, e.g., Di Pietro and Ern (2012) for details on how a similar criterion is expressed for the broken polynomial spaces used herein. Splitting the equations into momentum-guess and pressure-correction steps will in often make the system solvable also for equal-order discretisations, which typically do not satisfy inf–sup without additional stabilisation. That does not mean that there is no instability or loss of convergence order. See e.g. J.-L. Guermond and Quartapelle (1998) and Codina (2001) for more information. We have selected to use an inf–sup stable function-space pair in this work (quadratic polynomials for the velocity and linear for the pressure, P2P1), though the methods discussed are general and work for higher–order element pairs such as P3P2, and equal order such as P1P1 or P2P2—with the caveats about possible inf–sup instabilities and loss of convergence order in mind, especially for small time steps.

In addition to the pressure-correction methods discussed above, many other splitting methods for the Navier–Stokes equations exist, such as velocity-correction methods (J. Guermond and Shen 2003; Kawahara and Ohmiya 1985), and the more general Schur-complement methods (Schur 1917; Zhang 2005). The popular PISO (Issa 1986) and PIMPLE (Weller et al. 1998) algebraic pressure-correction techniques should also be mentioned, and we even implemented them in the Ocellaris flow solver that we have used in this work. They are exactly mass conserving, but unfortunately we were unable to make these converge with the expected order for our higher-order DG discretisation, so they have been left out of this comparison.

A prior comparison of algebraic and differential splitting methods in terms of divergence can be found in Quarteroni, Saleri, and Veneziani (2000). They compared differential and algebraic versions of the basic Chorin–Temam method, as well as introducing the algebraic quasi-compressible Yosida method and comparing it to a differential artificial compressibility method. They found that the algebraic methods satisfied the original incompressibility constraint exactly, while the differential methods did not. Satisfying the incompressibility constraint just as well as the coupled system does not mean that the resulting velocity fields were point-wise or element-wise divergence free, as the Yosida method

introduces compressibility and the elements used for all the calculations were the $P_1\text{iso}P_2/P_1$ elements, which are not locally divergence free even in the coupled case (Boffi et al. 2012).

In this paper we will look at the IPCS method on differential form in section II.3 and on algebraic form in section II.4. The DG SIMPLE method by Klein, Kummer, Keil, et al. (2015), Klein, Kummer, and Oberlack (2013), and Klein, Müller, et al. (2016) based on the SIMPLE method by Patankar and Spalding (1972) is described in section II.5. The effect of the choice of pressure-splitting scheme on the divergence of the resulting velocity field is discussed in section II.6, before the methods are compared on several benchmark cases in section II.7. The DG FEM spatial discretisation and the temporal scheme used in the numerical experiments are presented in section II.2. The results in this paper are not dependent on the details of the numerical scheme. The findings will hold for other methods, as long as discontinuous elements are used for the pressure and exact incompressibility is achieved when the governing equations are solved without pressure-velocity splitting. Exact incompressibility implies exact mass conservation for incompressible flow, and requires that the velocity field is pointwise divergence free inside each mesh cell and that the normal velocity is single valued across internal mesh facets. The volume fluxes across each cell's facets will then sum to zero for all cells in the mesh.

II.2 The DG FEM discretisation

This section contains a brief summary of the DG FEM discretisation of the governing equations (II.1) used to construct the block matrix in equation (II.2). More details can be found in Landet, Mardal, and Mortensen (2020) which is based on Cockburn, Kanschat, and Schötzau (2005), except for the elliptic term where we have used the symmetric interior-penalty (SIP) method by Arnold (1982), and not the local discontinuous Galerkin (LDG). Using SIP in this way has also been done by Shahbazi, Fischer, and Ethier (2007) and Cockburn, Kanschat, and Schötzau (2007). The exact details of the presented discretisation are *not* important for the results of this paper. The numerical scheme can be substituted for another numerical scheme as long as (i) the resulting mass matrix is block diagonal, (ii) the pressure is discontinuous and requires continuity-enforcing stabilisation of elliptic operators, and (iii) the velocity field is machine-precision divergence-free when equation (II.2) is solved on coupled form by a direct solver.

The Navier–Stokes equations from equation (II.1) are first discretised in time by use of values from previous time steps, $(\boldsymbol{u}^n, \boldsymbol{u}^{n-1})$, along with a suitable choice of time stepping coefficients $(\gamma_1, \gamma_2, \gamma_3)$. The time steps are $t = n\Delta t$ with initial conditions at $t = 0$. We have used a second-order backwards-differencing formulation, BDF2, where the coefficients are $(\gamma_1, \gamma_2, \gamma_3) = (3/2, -2, 1/2)$. If the initial value at $t = -\Delta t$ is not provided then coefficients $(1, -1, 0)$ are used for the first time step. The non-linearity of the convective term is handled semi-implicitly by introducing an explicit convecting velocity, $\boldsymbol{w} = 2\boldsymbol{u}^n - \boldsymbol{u}^{n-1}$.

The spatial differential equations are then

$$\rho\left(\frac{1}{\Delta t}(\gamma_1\boldsymbol{u}+\gamma_2\boldsymbol{u}^n+\gamma_3\boldsymbol{u}^{n-1})+(\boldsymbol{w}\cdot\nabla)\boldsymbol{u}\right)=\nabla\cdot\mu\left(\nabla\boldsymbol{u}+(\nabla\boldsymbol{u})^T\right)-\nabla p, \tag{II.4}$$

$$\nabla\cdot\boldsymbol{u}=0.$$

Let $\mathcal{T}$ be the set of all cells in the mesh and $\mathcal{S}$ the set of all facets. Polynomial function spaces of degree k on each cell K are denoted $P_k(K)$. These have no continuity at cell boundaries and no inherent boundary conditions. We use calligraphic typeface to denote operators and sets, bold italic for vectors functions and italic for scalar functions. Equation (II.4) is cast into the following form: find $\boldsymbol{u}\in[P_2(K)]^D$ and $p\in P_1(K)$ such that

$$\mathcal{A}(\boldsymbol{u},\boldsymbol{v};\boldsymbol{w})+\mathcal{B}(p,\boldsymbol{v})=\mathcal{D}(\boldsymbol{v})\quad\forall\ \boldsymbol{v}\ \in[P_2(K)]^D, \tag{II.5}$$

$$\mathcal{C}(\boldsymbol{u},q)=\mathcal{E}(q)\quad\forall\ q\ \in P_1(K), \tag{II.6}$$

in the tessellated domain $\mathcal{T}$ subject to

$$\boldsymbol{u}=\boldsymbol{u}_D\quad\text{on the boundary facets, }\mathcal{S}_D\subset\mathcal{S}. \tag{II.7}$$

The discontinuous Galerkin method works by breaking integrals over the whole domain into a sum of integrals over each mesh cell $K\in\mathcal{T}$, and defining fluxes of the unknown functions between these cells. The average and jump operators across an internal facet between two cells K^+ and K^- are defined as

$$\{\{u\}\}=\frac{1}{2}(u^+ + u^-), \tag{II.8}$$

$$[\![\, u\,]\!]=u^+-u^-, \tag{II.9}$$

$$[\![\,\boldsymbol{u}\,]\!]_{\boldsymbol{n}}=\boldsymbol{u}^+\cdot\boldsymbol{n}^+ + \boldsymbol{u}^-\cdot\boldsymbol{n}^-. \tag{II.10}$$

The convecting velocity $\boldsymbol{w}$ is $\mathrm{H}^{\mathrm{div}}$-conforming, i.e. the flux is continuous, $[\![\,\boldsymbol{w}\,]\!]_{\boldsymbol{n}}=0$. For exterior facets, let the connected cell be denoted K^+ such that $\boldsymbol{n}^+\cdot[\![\,\boldsymbol{u}\,]\!]=\boldsymbol{n}^+\cdot\boldsymbol{u}^+=\boldsymbol{n}\cdot\boldsymbol{u}$. Take $\{\{u\}\}=u$ and otherwise let all K^- values be zero on the boundary facets.

The momentum equation is discretised using the SIP method for the elliptic term (Arnold 1982) and otherwise using the fluxes from Cockburn, Kanschat, and Schötzau (2005). The flux of pressure is $\hat{p}=\{\{p\}\}$ and the convective flux $\hat{\boldsymbol{u}}^{\boldsymbol{w}}$ is a pure upwind flux. After multiplication with $\boldsymbol{v}$ and integration over $\mathcal{T}$, followed by integration by parts and application of the SIP method to the viscosity; $\mathcal{A}$ can be found as the bilinear part containing $\boldsymbol{u}=\boldsymbol{u}^{n+1}$, $\mathcal{B}$ as the bilinear part containing p, and $\mathcal{D}$ as the linear part of

$$\int_{\mathcal{T}}\frac{\rho}{\Delta t}(\gamma_1\boldsymbol{u}+\gamma_2\boldsymbol{u}^n+\gamma_3\boldsymbol{u}^{n-1})\boldsymbol{v}\,\mathrm{d}\boldsymbol{x} \tag{II.11}$$

$$-\int_{\mathcal{T}}\boldsymbol{u}\cdot\nabla\cdot(\rho\boldsymbol{v}\otimes\boldsymbol{w})\,\mathrm{d}\boldsymbol{x}\ +\ \int_{\mathcal{S}}\boldsymbol{w}\cdot\boldsymbol{n}^+\,\hat{\boldsymbol{u}}^{\boldsymbol{w}}\cdot[\![\,\rho\boldsymbol{v}\,]\!]\,\mathrm{d}s$$

$$
\begin{aligned}
&+ \int_{\mathcal{T}} \mu \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right) : \nabla \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \;+\; \int_{\mathcal{S}_I} \kappa_\mu [\![\, \boldsymbol{u} \,]\!] \cdot [\![\, \boldsymbol{v} \,]\!] \,\mathrm{d}s \\
&- \int_{\mathcal{S}} (\{\{\mu \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right)\}\} \cdot \boldsymbol{n}^+) \cdot [\![\, \boldsymbol{v} \,]\!] \,\mathrm{d}s \\
&- \int_{\mathcal{S}_I} (\{\{\mu \left(\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T\right)\}\} \cdot \boldsymbol{n}^+) \cdot [\![\, \boldsymbol{u} \,]\!] \,\mathrm{d}s \\
&- \int_{\mathcal{T}} p \nabla \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \;+\; \int_{\mathcal{S}} \hat{p}\, \boldsymbol{n}^+ \cdot [\![\, \boldsymbol{v} \,]\!] \,\mathrm{d}s + \frac{1}{2} \int_{\mathcal{T}} (\nabla \cdot \boldsymbol{w}) \boldsymbol{u} \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} = 0.
\end{aligned}
$$

On Dirichlet boundaries $\hat{\boldsymbol{u}}^{\boldsymbol{w}}$ is the upwind value. Depending on flow direction this is either $\boldsymbol{u}$ or $\boldsymbol{u}_D$. On all boundaries take $\hat{p} = p$. The SIP penalty parameter is selected based on Epshteyn and Rivière (2007) and Shahbazi, Fischer, and Ethier (2007),

$$
\kappa_\mu = 3 \frac{\mu_{\max}^2}{\mu_{\min}} k(k+1) \max_K \left(\frac{S_K}{V_K}\right), \tag{II.12}
$$

where the k is the polynomial order of the approximating polynomials, S_K is the cell surface area and V_K the cell volume. The penalty parameter is doubled on external facets.

Since the IPCS-D scheme is not exactly divergence free, the standard skew symmetric term (Gresho 1991) is included in the weak form used for all the methods. The term is the last integral in equation (II.11),

$$
\frac{1}{2} \int_{\mathcal{T}} (\nabla \cdot \boldsymbol{w}) \boldsymbol{u} \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x}. \tag{II.13}
$$

The continuity equation is multiplied by q and integrated over $\mathcal{T}$. The flux of velocity is here $\hat{\boldsymbol{u}}^p = \{\{\boldsymbol{u}\}\}$. After integration by parts $\mathcal{C}(\boldsymbol{u}, q)$ and $\mathcal{E}(q)$ can be found from

$$
\int_{\mathcal{S}} \hat{\boldsymbol{u}}^p \cdot \boldsymbol{n}^+ [\![\, q \,]\!] \,\mathrm{d}s \;-\; \int_{\mathcal{T}} \boldsymbol{u} \cdot \nabla q \,\mathrm{d}\boldsymbol{x} \;=\; 0. \tag{II.14}
$$

The non-zero $\mathcal{E}(q)$ results from using the boundary conditions in the flux, $\hat{\boldsymbol{u}}^p = \boldsymbol{u}_D$ on Dirichlet boundaries.

After solving equations (II.5) and (II.6) by one of the presented pressure-splitting schemes, the resulting velocity field is projected into a space where it is exactly divergence free. The resulting velocity field is pointwise divergence free inside each cell and the normal velocities across internal facets are continuous. This projection from weak to strong incompressibility requires only that the discrete version of the weak incompressibility criterion on the form shown in equation (II.14). The projection is described in Cockburn, Kanschat, and Schötzau (2005). It is not necessary to perform the projection inside the pressure iteration loops, it is run once as a post processing operation at the end of each time step. This BDM-like projection is local to each cell, so the result is computed by a dense solver on an element-by-element basis, see the paper by

Cockburn et al. for details showing that the projection is well-conditioned and results in machine precision pointwise divergence-free velocities. The results from the projection is used to compute the convecting velocities for the next time step, and also in the time integration; i.e. the un-projected velocity is not used afterwards.

II.2.1 Differential Poisson equation for the pressure

In the IPCS-D method, which will be described in section II.3, a Poisson equation must be solved for the pressure,

$$\nabla \cdot \left(\frac{1}{\rho} \nabla p \right) = f. \tag{II.15}$$

This equation is discretised with the SIP method, and the same penalty parameter as in equation (II.12) is used, with μ replaced by ρ^{-1}. The applied boundary condition is

$$\nabla p \cdot \boldsymbol{n} = 0 \tag{II.16}$$

on all boundaries. This is due to using Dirichlet boundary conditions for the velocity everywhere. See, e.g., Gresho and Sani (1987), and the references in the introduction to this work for more information related to the boundary conditions for the pressure-Poisson equation.

After multiplying by the test function $q \in P_1[K]$, integrating over the domain, performing integration by parts, and including the SIP stabilisation terms, the result is

$$\begin{aligned} &- \int_{\mathcal{T}} \nabla p \cdot \nabla q \, \mathrm{d}\boldsymbol{x} + \int_{\mathcal{S}_I} q \nabla p \cdot \boldsymbol{n} \, \mathrm{d}s \\ &+ \int_{\mathcal{S}_I} [\![q]\!] \{\{\nabla p\}\} \cdot \boldsymbol{n}^+ \, \mathrm{d}s + \int_{\mathcal{S}_I} [\![\nabla p]\!] \{\{q\}\} \cdot \boldsymbol{n}^+ \, \mathrm{d}s \\ &+ \int_{\mathcal{S}_I} [\![p]\!] \{\{\nabla q\}\} \cdot \boldsymbol{n}^+ \, \mathrm{d}s - \int_{\mathcal{S}_I} \kappa_p [\![p]\!] [\![q]\!] \, \mathrm{d}s = \int_{\mathcal{T}} f q \, \mathrm{d}\boldsymbol{x}, \end{aligned} \tag{II.17}$$

where $\mathcal{S}_I$ is the set of all internal facets. The null space due to using pure Neumann boundary conditions is removed in the Krylov solver by providing the basis of the null space along with the system matrix (Balay et al. 2014). The right-hand side will be the divergence of a known velocity field. This will be discretised as shown in equation (II.14) to minimise the differences between the differential and the algebraic methods.

The main difference between solving for the pressure this way, by a forming an elliptic differential equation, instead of performing the elliptic-like algebraic operation used in the IPCS-A method described in section II.3, is the introduction of the penalty parameter. The need for stabilisation of elliptic operators in DG methods means that the resulting discrete elliptic matrices are different between the algebraic and the differential methods. The consequences of this are explored in section II.6 and shown numerically in section II.7.

II.3 The IPCS-D method

The incremental pressure-correction scheme on differential form (IPCS-D) is an iterative version of the classic Chorin–Temam pressure correction scheme (Chorin 1968; Goda 1979; Temam 1969). The method begins with solving the momentum equation for an approximate velocity field $\boldsymbol{u}^*$ by use of a guessed pressure field, p^*,

$$\rho\left(\frac{1}{\Delta t}(\gamma_1\boldsymbol{u}^* + \gamma_2\boldsymbol{u}^n + \gamma_3\boldsymbol{u}^{n-1}) + (\boldsymbol{w}\cdot\nabla)\boldsymbol{u}^*\right) = \nabla\cdot\mu\left(\nabla\boldsymbol{u}^* + (\nabla\boldsymbol{u}^*)^T\right) - \nabla p^*. \tag{II.18}$$

A splitting error is now introduced. Make the assumption that the true velocity $\boldsymbol{u}$ and the velocity estimate $\boldsymbol{u}^*$ are close enough so that

$$(\boldsymbol{w}\cdot\nabla)(\boldsymbol{u}-\boldsymbol{u}^*) \approx 0 \tag{II.19}$$

$$\nabla\cdot\mu\left(\nabla(\boldsymbol{u}-\boldsymbol{u}^*) + (\nabla(\boldsymbol{u}-\boldsymbol{u}^*))^T\right) \approx 0. \tag{II.20}$$

By subtracting equation (II.18) from the true momentum equation (II.4) and using the assumptions in equations (II.19) and (II.20) the result is

$$\frac{1}{\Delta t}(\gamma_1\boldsymbol{u} - \gamma_1\boldsymbol{u}^*) = -\frac{1}{\rho}\nabla(p-p^*). \tag{II.21}$$

The velocity field should be divergence free, but the velocity estimate $\boldsymbol{u}^*$ is not due to having used an approximate pressure field p^*, and not the true velocity field p. The continuity equation, $\nabla\cdot\boldsymbol{u} = 0$, is now used to eliminate the unknown velocity field $\boldsymbol{u}$ from equation (II.21) by taking the divergence,

$$\nabla\cdot\left(\frac{\Delta t}{\gamma_1\rho}\nabla(p-p^*)\right) = \nabla\cdot\boldsymbol{u}^*, \tag{II.22}$$

which gives a Poisson equation for the unknown pressure p with the guessed pressure p^* and the velocity estimate $\boldsymbol{u}^*$ as known coefficients. When the pressure p is found it can be put back into equation (II.21) to find the updated velocity field, $\boldsymbol{u}$.

The IPCS-D algorithm

1. Guess p^* and solve equation (II.18) for $\boldsymbol{u}^*$.
2. Solve equation (II.22) for p.
3. Compute the updated velocity $\boldsymbol{u}$ from equation (II.21).
4. Check for convergence by computing the residual $r_u = ||\mathbf{u}^* - \mathbf{u}||$ and optionally go to step 1 using p as a new guess for the pressure if r_u is not sufficiently small.

II.4 The IPCS-A method

The incremental pressure-correction scheme on algebraic form (IPCS-A) starts from the Navier–Stokes equations on block-matrix form, as can be seen in equation (II.2). The momentum equation can be solved for an approximate velocity field $\mathbf{u}^*$ by use of a guessed pressure field $\mathbf{p}^*$,

$$\mathbf{A}\mathbf{u}^* = \mathbf{d} - \mathbf{B}\mathbf{p}^*. \tag{II.23}$$

A splitting error is now introduced. First let $\mathbf{A} = \mathbf{M} + \mathbf{R}$ where $\mathbf{M}$ is a scaled mass matrix resulting from the assembly of the first term in equation (II.3), and $\mathbf{R}$ contains the convective and diffusive operators. Then make the assumption that $\mathbf{R}(\mathbf{u} - \mathbf{u}^*) \approx 0$ and use this when subtracting equation (II.23) from the first line of equation (II.2), giving

$$\mathbf{M}(\mathbf{u} - \mathbf{u}^*) = -\mathbf{B}(\mathbf{p} - \mathbf{p}^*). \tag{II.24}$$

The matrix $\mathbf{M}$ is block diagonal and can easily be inverted. Use this property and the divergence free criterion, $\mathbf{C}\mathbf{u} = \mathbf{e}$, to remove the unknown $\mathbf{u}$ from equation (II.24)

$$\mathbf{e} - \mathbf{C}\mathbf{u}^* = -\mathbf{C}\mathbf{M}^{-1}\mathbf{B}(\mathbf{p} - \mathbf{p}^*). \tag{II.25}$$

Reorganising equation (II.25) results in an equation for $\mathbf{p}$

$$\mathbf{C}\mathbf{M}^{-1}\mathbf{B}\mathbf{p} = \mathbf{C}\mathbf{M}^{-1}\mathbf{B}\mathbf{p}^* - \mathbf{e} + \mathbf{C}\mathbf{u}^*. \tag{II.26}$$

The velocity $\mathbf{u}$ can be recovered without solving a linear system, simply by substituting the pressure $\mathbf{p}$ from the solution of equation (II.26) into equation (II.24),

$$\mathbf{u} = -\mathbf{M}^{-1}\mathbf{B}(\mathbf{p} - \mathbf{p}^*). \tag{II.27}$$

The IPCS-A algorithm

1. Guess $\mathbf{p}^*$ and solve equation (II.23) for $\mathbf{u}^*$.

2. Solve equation (II.26) for $\mathbf{p}$.

3. Compute the updated velocity $\mathbf{u}$ from equation (II.27).

4. Check for convergence by computing the residual $r_u = ||\mathbf{u}^* - \mathbf{u}||$ and optionally go to step 1 using $\mathbf{p}$ as a new guess for the pressure if r_u is not sufficiently small.

II.5 The SIMPLE method

This description is a summary of the method presented in Klein, Kummer, and Oberlack (2013), the semi-implicit method for pressure-linked equations, SIMPLE. The method starts with the Navier–Stokes equations on algebraic block matrix form, equation (II.2). A guess $\mathbf{p}^*$ is made and inserted into the governing equations which can be solved for an estimated velocity field, $\mathbf{u}^*$,

$$\mathbf{A}\mathbf{u}^* = \mathbf{d} - \mathbf{B}\mathbf{p}^*, \tag{II.28}$$

$$\mathbf{C}\mathbf{u}^* = \mathbf{e} + \mathbf{e}^*, \tag{II.29}$$

where $\mathbf{e}^*$ is not zero when $\mathbf{p}^*$ is not a perfect guess due to $\nabla \cdot \boldsymbol{u}^* \neq 0$. Subtracting these equations from equation (II.2) and defining corrections $\hat{\mathbf{u}} = \mathbf{u} - \mathbf{u}^*$ and $\hat{\mathbf{p}} = \mathbf{p} - \mathbf{p}^*$ gives

$$\mathbf{A}\hat{\mathbf{u}} = -\mathbf{B}\hat{\mathbf{p}}, \tag{II.30}$$

$$\mathbf{C}\hat{\mathbf{u}} = -\mathbf{e}^*. \tag{II.31}$$

A splitting error is now introduced. Construct a matrix $\tilde{\mathbf{A}} \approx \mathbf{A}$ where $\tilde{\mathbf{A}}$ approximates $\mathbf{A}$, but is much easier to invert. A diagonal or block diagonal version of $\mathbf{A}$ is used as an approximation in the numerical examples below. Note that this approximation allows the time derivative to be removed for steady-state problems. This is not true for the IPCS-A method, where we must require $\mathbf{M} \neq 0$. The velocity correction in equation (II.30) can then be approximated as

$$\hat{\mathbf{u}} = -\tilde{\mathbf{A}}^{-1}\mathbf{B}\hat{\mathbf{p}}, \tag{II.32}$$

and this approximation can be used to solve for $\hat{\mathbf{p}}$ by substitution into equation (II.31)

$$\left[\mathbf{C}\tilde{\mathbf{A}}^{-1}\mathbf{B}\right]\hat{\mathbf{p}} = \mathbf{e}^* = \mathbf{C}\mathbf{u}^* - \mathbf{e}. \tag{II.33}$$

The SIMPLE method is not guaranteed to converge without under-relaxation in the updates of $\mathbf{p}$ and $\mathbf{u}$ due to the approximate $\tilde{\mathbf{A}}$. The under-relaxation of the pressure is performed explicitly,

$$\mathbf{p} = \mathbf{p}^* + \alpha_p \hat{\mathbf{p}}, \tag{II.34}$$

while an implicit scheme is used for the velocity,

$$\left[\frac{1-\alpha_u}{\alpha_u}\tilde{\mathbf{A}} + \mathbf{A}\right]\mathbf{u}^* = \mathbf{d} - \mathbf{B}\mathbf{p}^* + \left[\frac{1-\alpha_u}{\alpha_u}\tilde{\mathbf{A}}\right]\mathbf{u}^*_{\text{prev}}, \tag{II.35}$$

with $0 < (\alpha_p, \alpha_u) < 1$.

The SIMPLE algorithm

1. Solve for $\mathbf{u}^*$ using equation (II.35) starting with guesses $\mathbf{u}^*_{\text{prev}}$ and $\mathbf{p}^*$

2. Find $\hat{\mathbf{p}}$ using equation (II.33)
3. Update $\mathbf{p}$ using equation (II.34)
4. Update $\mathbf{u}$ using equation (II.32)
5. Check for convergence by computing the residual $r_u = ||\mathbf{u}^* - \mathbf{u}||$ and optionally go to step 1 with new guesses for $\mathbf{u}^*$ and $\mathbf{p}^*$

A note on constructing $\tilde{\mathbf{A}}$: It is natural to think that the closer $\tilde{\mathbf{A}}$ is to $\mathbf{A}$, while still being easy to invert, the more accurate each iteration will be. A more accurate approximation should hence result in fewer iterations for a given tolerance. We have tried with a block-diagonal matrix $\tilde{\mathbf{A}}$, using a dense linear algebra library to invert the single-element dense blocks, and also tested bundling neighbouring elements into multi-element dense blocks. As long as these blocks are not overly large, the time to construct $\tilde{\mathbf{A}}^{-1}$ will not dominate the iteration procedure, as this is done once per time step. What we found is that contrary to the intuition, and fully in line with Klein, Kummer, and Oberlack (2013), the closer the matrix $\tilde{\mathbf{A}}$ is to the identity matrix, the more efficient the iteration procedure becomes. Results comparing a diagonal and a block-diagonal $\tilde{\mathbf{A}}$ matrix are shown in figure II.1. In these results only one element is included in each block, but the trend is the same when including more, the diagonal version converges faster.

II.6 Exact mass conservation

The numerical representation of the velocity field will be divergence free to the precision of the discrete operators if $\mathbf{Cu} - \mathbf{e} = 0$. In the IPCS-A method this is ensured on the algebraic level,

$$\begin{aligned} \mathbf{Cu} - \mathbf{e} = \; & \mathbf{Cu} - \mathbf{Cu}^* + \mathbf{Cu}^* - \mathbf{e} \\ \overset{(II.24)}{=} \; & \mathbf{CM}^{-1}\mathbf{B}(\mathbf{p} - \mathbf{p}^*) + \mathbf{Cu}^* - \mathbf{e} \\ \overset{(II.25)}{=} \; & 0. \end{aligned} \tag{II.36}$$

The IPCS-D pressure Poisson equation (II.22) is not equivalent to equation (II.25) due to the stabilisation jump terms which will introduce a residual divergence in the velocity field, even if the boundary conditions applied to equation (II.22) are in perfect agreement with equation (II.26). The reason is that the assembled matrix from the elliptic operator with stabilisation does not satisfy equation (II.25),

$$\text{Assemble}_{\text{DG},\kappa_p}\left(\nabla \cdot (\frac{\Delta t}{\gamma_1 \rho}\nabla \bullet)\right) \neq \mathbf{CM}^{-1}\mathbf{B}. \tag{II.37}$$

Other elliptic DG FEM discretisations like LDG and NIPG will also contain stabilisation terms (Arnold et al. 2002), and will hence have the same problem.

A similar argument is shown in Klein, Kummer, Keil, et al. (2015). They also show that the SIMPLE method is exactly divergence free with an argument similar to equation (II.36).

II.7 Numerical experiments

The weak form described in section II.2 is implemented with quadratic discontinuous Galerkin elements for the velocity (DG2) and linear DG elements for the pressure (DG1). The numerical experiments are run with iterative solvers from PETSc 3.8 (Balay et al. 2014). All results are from simulations running in parallel with MPI on the Abel computational cluster at the University of Oslo. The GMRES iterative solver has been used for both velocities and pressures. The relative and absolute error criteria are both 1×10^{-15} with a maximum of 100 Krylov iterations. Convergence in the Krylov solver is typically achieved in less than 100 Krylov iterations for all but the first pressure-correction iteration in the first time step.

The additive-Schwarz method (ASM) is used to precondition the velocity solvers and for the pressure an algebraic multigrid (AMG) preconditioner is applied, we use HYPRE BoomerAMG. The solvers are set up to use the previous solution as the initial guess. The additive-Schwarz preconditioner uses PETSc's default settings, which is incomplete LU factorization on each CPU. The BoomerAMG preconditioner also uses PETSc's default settings.

All the results shown below have been produced by Ocellaris (Landet 2019b; Landet 2019c), a single- and multiphase DG FEM Navier–Stokes solver. Input files for Ocellaris and post-processing scripts, which can be used to reproduce all figures shown in this paper, can be found on Zenodo in Landet (2019a).

II.7.1 Taylor–Green 2D flow

The Taylor–Green vortex is an analytical solution to the 2D incompressible Navier–Stokes equations. The solution is

$$\begin{aligned} u &= -\sin(\pi y)\cos(\pi x)\exp(-2\pi^2\nu t), \\ v &= \sin(\pi x)\cos(\pi y)\exp(-2\pi^2\nu t), \\ p &= -{}^1\!/_4\,\rho(\cos 2\pi x + \cos 2\pi y)\exp(-4\pi^2\nu t). \end{aligned} \tag{II.38}$$

Spatial convergence of the pressure-correction methods is examined on a regular grid. The domain, $(x, y) \in [0, 2] \times [0, 2]$, is divided into $N \times N$ rectangles that are then subdivided into two triangles each. Grid sizes $N = \{8, 16, 24, 32\}$ are used to study the spatial convergence rate. The time step is $\Delta t = 0.01$ and the numerical results are compared to the analytical expressions at $t = 1.0$. Exactly 160 pressure-correction iterations are performed for each time step, which is sufficient for all methods, see figure II.1. The physical parameters applied are $\rho = 1.0$ and $\nu = 0.005$.

In the SIMPLE method an under-relaxation factor of $\alpha_u = 0.7$ is used for the velocity and $\alpha_p = 1.0$ is used for the pressure. This is in line with Klein,

Kummer, and Oberlack (2013). Setting both under-relaxation factors to 1.0 (no under-relaxation) as was done in Klein, Kummer, Keil, et al. (2015) is unstable for the current simulations and causes the iteration procedure to blow up. A diagonal $\tilde{\mathbf{A}}$ matrix is used with no lumping.

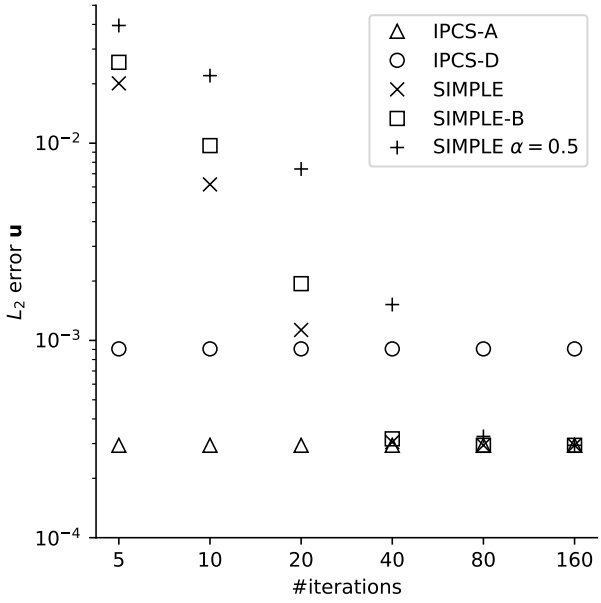
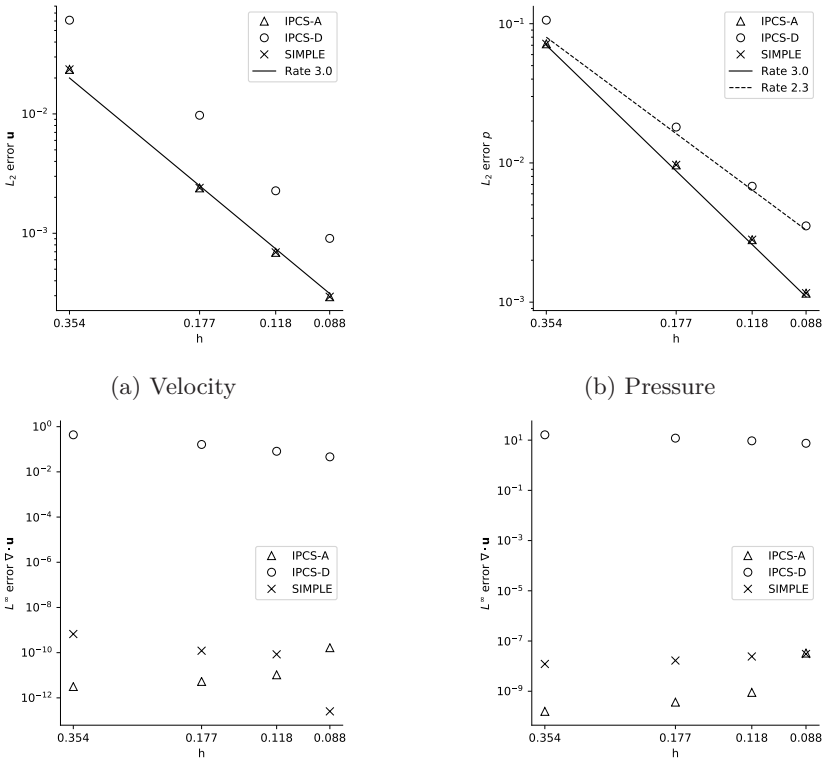


Figure II.1: Taylor–Green test case: effect of the number of pressure-correction iterations per time step. The SIMPLE method is tested with a block-diagonal $\tilde{\mathbf{A}}$ matrix (one block for each cell, 'SIMPLE-B') in addition the the diagonal $\tilde{\mathbf{A}}$ used for the rest of the simulations. A set of simulations with under-relaxation parameters $(\alpha_u, \alpha_p) = (0.5, 0.5)$ is included in addition to the $(\alpha_u, \alpha_p) = (0.7, 1.0)$ choice used for the rest of the simulations.

The number of pressure-correction iterations required to reach convergence is shown in figure II.1. The SIMPLE method requires far more iterations in order to converge than the IPCS methods. The influence of some of the tunable parameters in the SIMPLE method are also shown in the figure. Some sensitivity is found both in regard to how the approximate inverse of $\mathbf{A}$ is computed, and in the choice of under-relaxation parameters. The effect of these choices are significantly smaller than the effect of changing the pressure-correction method.

Spatial convergence rates of the velocity and pressure can be seen in figures II.2a and II.2b. The velocity converges with the expected order while the pressure super-converges due to the regular mesh. It is normal to observe super-convergence on highly regular meshes (Guillén-González and Tierra 2012). The divergence of the velocity field is shown in figure II.2c. The divergence has been projected to a piecewise constant (DG0) function space before computing

the L_2 norm. The IPCS-D method gives a velocity field with approximately ten orders of magnitude higher divergence than IPCS-A and SIMPLE. The divergence is also computed in the space of the velocity, piecewise quadratics (DG2), and this is shown in figure II.2d. The same behaviour is shown here, though the error is higher.



(a) Velocity

(b) Pressure

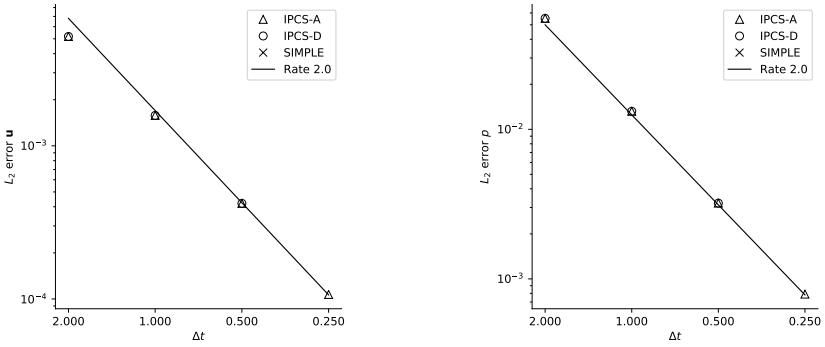
(c) Divergence of $\boldsymbol{u}$ in DG0

(d) Divergence of $\boldsymbol{u}$ in DG2

Figure II.2: Taylor–Green test case: spatial convergence.

Studying temporal convergence for the selected Taylor–Green flow requires a very fine spatial discretisation since the error in the time stepping routine is much smaller than the spatial error. The results obtained with a very fine grid where $N = 200$ is shown in figure II.3. The time step is gradually refined with $\Delta t = \{2.00, 1.00, 0.50, 0.25\}$, and the numerical solution is compared with the analytical at $t = 6.0$. The number of pressure-correction iterations per time step is 200 and the number of Krylov solver iterations is also set to 200. This is a necessary increase to achieve the theoretical convergence rate for the velocity. The pressure will converge at a rate slightly above 2.0 with fewer iterations, but even with the fine spatial discretisation, the convergence rate of the velocity is slightly below 2.0 and the Krylov solver is never fully converged even after

200 GMRES iterations per pressure-correction iteration. Between refinements $\Delta t = 0.5$ and $\Delta t = 0.25$, the observed convergence rate of the velocity is 1.97.



(a) Velocity (b) Pressure

Figure II.3: Taylor–Green test case: temporal convergence.

II.7.2 Ethier–Steinman 3D flow

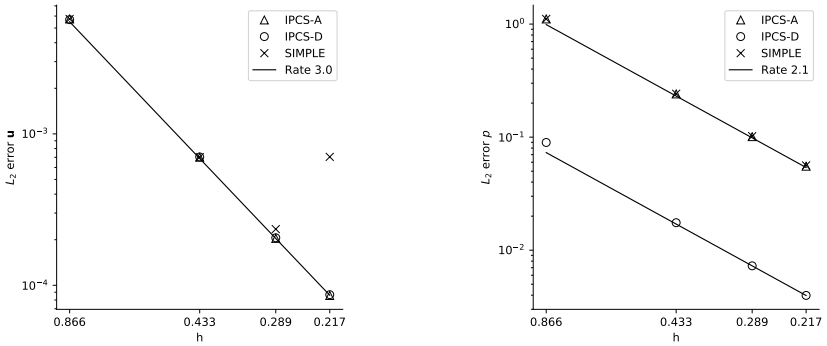
An analytical solution to the 3D incompressible Navier–Stokes equations is given by Ethier and Steinman (1994, equation 15) as

$$\begin{aligned}
u &= -ae^{-b^2\nu t}[e^{ax}\sin(ay+bz) + e^{az}\cos(ax+by)],\\
v &= -ae^{-b^2\nu t}[e^{ay}\sin(az+bx) + e^{ax}\cos(ay+bz)],\\
v &= -ae^{-b^2\nu t}[e^{az}\sin(ax+by) + e^{ay}\cos(az+bx)],\\
p &= \frac{-a^2}{2}e^{-2b^2\nu t}[e^{2ax} + e^{2ay} + e^{2az} - \bar{p}\\
&\qquad + 2\sin(ax+by)\cos(az+bx)e^{a(y+z)}\\
&\qquad + 2\sin(ay+bz)\cos(ax+by)e^{a(z+x)}\\
&\qquad + 2\sin(az+bx)\cos(ay+bz)e^{a(x+y)}].
\end{aligned} \tag{II.39}$$

A constant correction for the pressure, $\bar{p}$, is included in equation (II.39) to ensure that the average pressure is zero. This is enforced in the linear solver to remove the pressure kernel from the system. On a domain $(x, y, z) \in [-1, 1] \times [-1, 1] \times [-1, 1]$ this gives

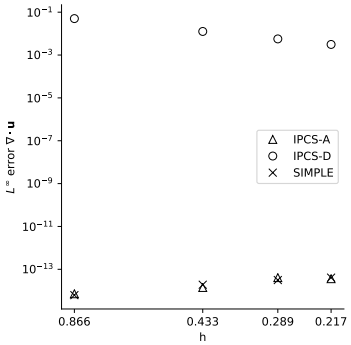
$$\bar{p} = [-15\pi^2 - 32 + 32e^{\pi} + 15\pi^2 e^{\pi}](e^{\frac{\pi}{2}}5\pi^3)^{-1}. \tag{II.40}$$

Dirichlet boundary conditions are applied to the velocity. The domain is divided into a regular mesh with $N \times N \times N$ cubes which are further subdivided into six tetrahedra in each cube. The time step is $\Delta t = 0.001$ and the numerical solution is compared to the analytical at time $t = 0.1$.



(a) Velocity

(b) Pressure



(c) Divergence of $\boldsymbol{u}$ in DG0

Figure II.4: Ethier–Steinman test case: spatial convergence.

The spatial convergence of the methods in Ethier–Steinman flow can be seen in figure II.4. The results for the IPCS methods are presented with 100 pressure-correction iterations per time step, but the results differ only in the third significant digit from the results obtained with only 5 iterations. The SIMPLE method has been run with $\alpha_u = \alpha_p = 0.5$ to get stable iterations at the finer grid resolutions where using $\alpha_u = 0.7$ and $\alpha_p = 1.0$ would blow up. The SIMPLE results are produced using 500 pressure-correction iterations per time step. This gives an optimal convergence rate for the pressure at all tested mesh resolutions and for the velocity at all but the finest resolution. With 100 pressure-correction iterations, only two points would fall on the line in figure II.4a. The divergence error norms for the algebraic methods are again close to machine precision. The IPCS-D divergence error is much higher, but getting smaller with increasing mesh resolution, as expected.

II.7.3 Efficiency

Producing a fair wall-clock comparison of the run time of the pressure correction methods is more tricky than reporting convergence results. For the convergence rates, it only matters that the Krylov iterations inside each pressure-correction iteration converge in the end, not how many iterations are needed for convergence. The total running time is totally dominated by the time spent in the momentum-prediction and pressure-correction steps, and except for matrix assembly, which takes around 1% of the wall clock time, this is all time spent in the Krylov solvers. Tweaking the preconditioner parameters used for the Krylov solvers in each method may improve one of the methods more than the other, causing this comparison—which uses the same preconditioner parameters for all three methods—to not be very relevant. With that caveat in mind, the time spent in the two steps can be seen in figure II.5.

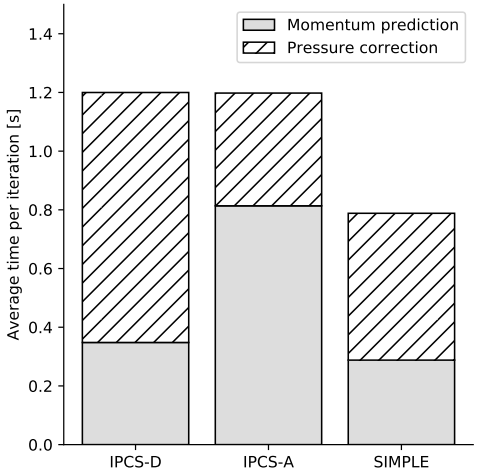


Figure II.5: The average wall clock time spent in the two pressure correction steps. Ethier–Steinman flow, N=12, 100 pressure corrections in each of the 100 time steps, 16 CPUs. The minimum average times from five full simulations of each method have been used. The noticeable differences between the three methods are very repeatable.

The total time per pressure-correction iteration is less in the SIMPLE method than in the IPCS methods, which is to be expected due to the under-relaxation procedure causing the unknown vectors to be closer to the initial guesses. The IPCS-D method spends more time in the pressure-correction than the IPCS-A method. The reason may be that the penalty parameter, which ensures coercivity, also makes the resulting matrix more and more ill-conditioned as the penalty parameter increases, although the parameter we have used, equation (II.12), should be close to optimal. The momentum-prediction step takes significantly more time in IPCS-A than in IPCS-D. The reason may be related to the skew

symmetric convective term, equation (II.13), making the IPCS-D **A** matrix more diagonal dominant. Unfavourable right-hand sides may also require more iterations.

It should be noted that the results in figure II.5 must be seen in light of figure II.1. The SIMPLE method requires far more iterations to achieve convergence, so while it is faster per iteration it is much slower if the metric is time to solution and not time per iteration.

II.8 Conclusions

This paper compares three existing pressure correction methods, two algebraic and one on differential form. We show algebraically why the algebraic pressure correction methods retain the machine-precision-zero divergence at the end of each iteration, while differential methods of formulating the pressure-Poisson equation in DG FEM must include stabilising terms that cause a residual divergence. This error only disappears in the limit of an exactly resolved solution where there are no longer any inter-element jumps. For practical purposes this will never happen.

For both the 2D (Taylor–Green, figure II.2) and 3D (Ethier–Steinman, figure II.4) numerical experiments, the convergence results show that the algebraic pressure-splitting methods are exactly divergence free, while the differential IPCS-D method is not. The IPCS methods converge perfectly with only five pressure-correction iterations per time step for both the 2D and the 3D test cases, while the SIMPLE method requires between 10 and 100 times more iterations to produce converged results. The SIMPLE method spends slightly less time per iteration (figure II.5), but is not fast enough to outweigh the vastly increased number of iterations required to obtain converged results. For mass conserving pressure-correction iterations, the IPCS-A method is most efficient, and hence the recommended option among the three tested methods.

Acknowledgements

The simulations were performed on resources provided by UNINETT Sigma2, the national infrastructure for high performance computing and data storage in Norway.

References

Amestoy, P. R., Duff, I. S., et al. (2001). “A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling”. *SIAM Journal on Matrix Analysis and Applications* 23.1, pp. 15–41.

Amestoy, P. R., Guermouche, A., et al. (2006). “Hybrid scheduling for the parallel solution of linear systems”. *Parallel Computing* 32.2, pp. 136–156.

Arnold, D. N. (1982). “An interior penalty finite element method with discontinuous elements”. *SIAM journal on numerical analysis* 19.4, pp. 742–760.

Arnold, D. N. et al. (2002). "Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems". *SIAM J. Numer. Anal.* 39.5, pp. 1749–1779.

Badia, S. and Codina, R. (2007). "Algebraic pressure segregation methods for the incompressible Navier–Stokes equations". *Archives of Computational Methods in Engineering* 15.3, pp. 1–52.

Balay, S. et al. (2014). *PETSc Users Manual*. ANL-95/11-Revision 3.5.

Boffi, D. et al. (2012). "Local Mass Conservation of Stokes Finite Elements". *Journal of Scientific Computing* 52.2, pp. 383–400.

Chorin, A. J. (1968). "Numerical solution of the Navier-Stokes equations". *Mathematics of computation* 22.104, pp. 745–762.

Cockburn, B., Kanschat, G., and Schötzau, D. (2005). "A locally conservative LDG method for the incompressible Navier-Stokes equations". *Mathematics of Computation* 74.251, pp. 1067–1096.

— (2007). "A Note on Discontinuous Galerkin Divergence-free Solutions of the Navier–Stokes Equations". *Journal of Scientific Computing* 31.1-2, pp. 61–73.

Codina, R. (2001). "Pressure Stability in Fractional Step Finite Element Methods for Incompressible Flows". *Journal of Computational Physics* 170.1, pp. 112–140.

Di Pietro, D. A. and Ern, A. (2012). *Mathematical Aspects of Discontinuous Galerkin Methods*. Springer.

Epshteyn, Y. and Rivière, B. (2007). "Estimation of penalty parameters for symmetric interior penalty Galerkin methods". *Journal of Computational and Applied Mathematics* 206.2, pp. 843–872.

Ethier, C. R. and Steinman, D. A. (1994). "Exact fully 3D Navier–Stokes solutions for benchmarking". *International Journal for Numerical Methods in Fluids*.

Fletcher, R. (1976). "Conjugate gradient methods for indefinite systems". In: *Numerical Analysis*. Ed. by Watson, G. A. Lecture Notes in Mathematics. Springer Berlin Heidelberg, pp. 73–89.

Goda, K. (1979). "A multistep technique with implicit difference schemes for calculating two- or three-dimensional cavity flows". *Journal of Computational Physics* 30.1, pp. 76–95.

Gresho, P. M. (1991). "Some current CFD issues relevant to the incompressible Navier-Stokes equations". *Computer Methods in Applied Mechanics and Engineering* 87.2, pp. 201–252.

Gresho, P. M. and Sani, R. L. (1987). "On pressure boundary conditions for the incompressible Navier-Stokes equations". *International Journal for Numerical Methods in Fluids* 7.10, pp. 1111–1145.

Guermond, J. L., Minev, P., and Shen, J. (2006). "An overview of projection methods for incompressible flows". *Computer Methods in Applied Mechanics and Engineering* 195.44, pp. 6011–6045.

Guermond, J.-L. and Quartapelle, L. (1998). "On stability and convergence of projection methods based on pressure Poisson equation". *International Journal for Numerical Methods in Fluids* 26.9, pp. 1039–1053.

Guermond, J. and Shen, J. (2003). "Velocity-Correction Projection Methods for Incompressible Flows". *SIAM Journal on Numerical Analysis* 41.1, pp. 112–134.

Guillén-González, F. and Tierra, G. (2012). "Superconvergence in velocity and pressure for the 3D time-dependent Navier-Stokes Equations". *SeMA Journal* 57.1, pp. 49–67.

Hestenes, M. R. and Stiefel, E. (1952). *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS.

Issa, R. (1986). "Solution of the implicitly discretised fluid flow equations by operator-splitting". *Journal of Computational Physics* 62.1, pp. 40–65.

Kan, J. van (1986). "A second-order accurate pressure-correction scheme for viscous incompressible flow". *SIAM journal on scientific and statistical computing* 7.3, pp. 870–891.

Karniadakis, G. E., Israeli, M., and Orszag, S. A. (1991). "High-order splitting methods for the incompressible Navier-Stokes equations". *Journal of Computational Physics* 97.2, pp. 414–443.

Kawahara, M. and Ohmiya, K. (1985). "Finite element analysis of density flow using the velocity correction method". *International Journal for Numerical Methods in Fluids* 5.11, pp. 981–993.

Klein, B., Kummer, F., Keil, M., et al. (Apr. 2015). "An extension of the SIMPLE based discontinuous Galerkin solver to unsteady incompressible flows". *International Journal for Numerical Methods in Fluids* 77.10, pp. 571–589.

Klein, B., Kummer, F., and Oberlack, M. (Mar. 2013). "A SIMPLE based discontinuous Galerkin solver for steady incompressible flows". *Journal of Computational Physics* 237, pp. 235–250.

Klein, B., Müller, B., et al. (July 2016). "A high-order discontinuous Galerkin solver for low Mach number flows". *International Journal for Numerical Methods in Fluids* 81.8, pp. 489–520.

Landet, T. (2019a). *Input files and plots*. Zenodo: 10.5281/zenodo.2556909.

— (2019b). "Ocellaris: a DG FEM solver for free-surface flows". *The Journal of Open Source Software* 4.35, p. 1239.

— (2019c). *The Ocellaris finite element solver for free surface flows*. Version 2019.0.1. www.ocellaris.org.

Landet, T., Mardal, K.-A., and Mortensen, M. (2020). "Slope limiting the velocity field in a discontinuous Galerkin divergence-free two-phase flow solver". *Computers & Fluids* 196, p. 104322.

Li, X. et al. (Sept. 1999). *SuperLU Users' Guide*. Tech. rep. LBNL-44289. crd.lbl.gov/~xiaoye/SuperLU. Lawrence Berkeley National Laboratory.

Li, X. S. (Sept. 2005). "An Overview of SuperLU: Algorithms, Implementation, and User Interface". *ACM Transactions on Mathematical Software* 31.3, pp. 302–325.

Li, X. S. and Demmel, J. W. (2003). "SuperLU_DIST: A Scalable Distributed-memory Sparse Direct Solver for Unsymmetric Linear Systems". *ACM Trans. Math. Softw.* 29.2, pp. 110–140.

Orszag, S. A., Israeli, M., and Deville, M. O. (1986). "Boundary conditions for incompressible flows". *Journal of Scientific Computing* 1.1, pp. 75–111.

Patankar, S. V. and Spalding, D. B. (1972). "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows". *International Journal of Heat and Mass Transfer* 15.10, pp. 1787–1806.

Quarteroni, A., Saleri, F., and Veneziani, A. (2000). "Factorization methods for the numerical approximation of Navier–Stokes equations". *Computer Methods in Applied Mechanics and Engineering* 188.1, pp. 505–526.

Saad, Y. and Schultz, M. (1986). "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems". *SIAM Journal on Scientific and Statistical Computing* 7.3, pp. 856–869.

Sani, R. L. et al. (2006). "Pressure boundary condition for the time-dependent incompressible Navier–Stokes equations". *International Journal for Numerical Methods in Fluids* 50.6, pp. 673–682.

Schur, I. (1917). "Über Potenzreihen, die im Innern des Einheitskreises beschränkt sind". *Journal für die reine und angewandte Mathematik* 147, pp. 205–232.

Shahbazi, K., Fischer, P. F., and Ethier, C. R. (2007). "A high-order discontinuous Galerkin method for the unsteady incompressible Navier–Stokes equations". *Journal of Computational Physics* 222.1, pp. 391–407.

Temam, R. (1969). "Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires (II)". *Archive for rational mechanics and analysis* 33.5, pp. 377–385.

Vorst, H. van der (1992). "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems". *SIAM Journal on Scientific and Statistical Computing* 13.2, pp. 631–644.

Weller, H. G. et al. (Nov. 1998). "A tensorial approach to computational continuum mechanics using object-oriented techniques". *Computers in Physics* 12.6, pp. 620–631.

Zhang, F., ed. (2005). *The Schur Complement and Its Applications*. Numerical Methods and Algorithms. Springer US.

III

Paper III

Ocellaris: a discontinuous Galerkin finite element solver for two-phase flows with high density differences

Tormod Landet, Mikael Mortensen

Abstract

In free-surface flows, such as breaking ocean waves, the momentum field will have a discontinuity at the interface between the two immiscible fluids, air and water, but still be smooth in most of the domain. Using a higher-order numerical method is more efficient than increasing the number of low-order computational cells in areas where the solution is smooth, but higher-order approximations cause convective instabilities at discontinuities. In Ocellaris we use slope limiting of discontinuous Galerkin solutions to stabilise finite element simulations of flows with large density jumps, which would otherwise blow up due to Gibbs oscillations resulting from approximating a factor 1000 sharp jump (air to water) by higher-order shape functions.

We have previously shown a slope-limiting procedure for velocity fields that is able to stabilise 2D free-surface simulations running on a single CPU. In this paper our solver is extended to 3D and coupled to an algebraic pressure-correction scheme that retains the exact incompressibility of the direct solution used in the 2D simulations. We have tested the method on a common 3D dam-breaking test case and compared the free-surface evolution and impact pressures to experimental results. We then include an existing forcing-zone approach in order to simulate a surface-piercing vertical cylinder in an infinite wave field. In both cases the free-surface elevation, forces, and pressures compare well with published experiments. The Ocellaris solver is available as an open-source and well-documented program along with the input files needed to replicate the included results (www.ocellaris.org).

III.1 Introduction

Increasing the polynomial approximation order is more efficient than increasing the number of computational cells when solving partial differential equations (PDEs) where the solution is smooth (Babuška and Dorr 1981). Performing more work locally in higher-order cells is also advantageous for today's parallel computers (Huerta et al. 2013; R. M. Kirby, Sherwin, and Cockburn 2012; Kubatko et al. 2009). However, for equations with convective operators, a jump in the solution or the coefficients will cause nonlinear convective instabilities—spurious Gibbs oscillations at discontinuities—that will eventually destroy the solution if the ratio between the high and low sides of the jump is large. Linear numerical schemes, such as the discontinuous Galerkin finite element method (DG FEM), produces linear algebraic equations from discretisation of linear PDEs. Such linear schemes cannot guarantee convective stability if they are not monotonic, and a linear monotonic scheme must be first-order and is hence highly dissipative (Harten 1983; Krivodonova et al. 2004). One way to get around the problem of having to chose between high-order approximations and convective stability is to make the scheme nonlinear by the inclusion of a nonlinear cell-wise projection operator, a slope limiter.

We have previously shown that component-wise slope limiters can be used on the convected velocity field to stabilise two-phase flow simulations with large density jumps (Landet, Mardal, and Mortensen 2020). Most existing methods used for the simulation of air/water two-phase flows employ low-order finite volume methods for the discretisation of the governing PDEs. In these methods the solution is piecewise constant, and the convective instabilities can be dealt with by using a flux limiter—a non-linear facet-local parameter that blends the upwind and downwind fluxes to obtain stability without excessive diffusion (Hirt and Nichols 1981; Kleefsman et al. 2005; Popinet 2003; Ubbink 1997; Weller et al. 1998).

Ocellaris solves the variable-density Navier–Stokes equations for two-phase flow (section III.2). The numerical method is based on a higher-order interior-penalty DG FEM (section III.3) with a component-wise velocity slope limiter for the convected velocity field (section III.3.4) and an algebraic pressure-correction method for the pressure–velocity coupling (section III.4). A regular wave model based on a stream-function formulation is used for initial and boundary values for water-wave simulations. The details of this model are presented in section III.5, including how the stream-function approach can be used to specify one velocity field, valid in both the air and water phases, which is divergence free, satisfies all boundary conditions, and conforms to the free surface. Section III.5 also describes how a forcing-zone approach is used for damping free-surface disturbances near the boundaries in order to avoid unwanted reflections. The implementation of Ocellaris is briefly described in section III.6, and the main steps required to set up an Ocellaris simulation are described in section III.6.1.

The `BlendedAlgebraicVOF` multiphase model used in this paper computes a piecewise constant density distribution based on an algebraic VOF method (Muzaferija et al. 1998). Other multiphase models are available, and it is also

possible to define a custom model in the Ocellaris simulation input file (YAML format, see Landet (2019b) for details). Using such a very simple model is not optimal, but the velocity slope limiter will lower the effective order of the obtained velocities at the density jump, so perhaps not much is lost in terms of obtainable accuracy from using a relatively simple free-surface capturing method. An interface-capturing method that can take full advantage of the high order of the convecting velocity field is an interesting research topic, and would most likely require fewer cells in the free-surface region than what is used here. Still, the results show good agreement with lab experiments, and away from the free surface the overall method retains the high-order approximation properties of the Navier–Stokes discretisation since the true density field is constant here.

Section III.7.1 shows the performance of the Ocellaris solver on a 3D dam-breaking test case. The test is meant to simulate a "green-water" event, a large wave breaking over a ship deck and impacting the cargo, in this case a single container outfitted with pressure gauges. The results show that both the free-surface evolution and the impact pressures compare well with published experiments. The second test case, presented in section III.7.2, is a vertical cylinder exposed to steep regular waves. This is meant to simulate wave loads on an offshore windmill or another type of slender surface-piercing marine structure. This test shows good agreement in regard to the total force on the cylinder when compared to laboratory experiments. The test also shows that the existing forcing-zone approach summarised in section III.5 works well in a DG FEM setting to dampen free-surface disturbances near the inlet and outlet of the numerical wave tank. Finally, the discussion in section III.8 concludes that Ocellaris' high-order multiphase flow solver can successfully simulate complex 3D air/water free-surface flows at high Reynolds numbers.

III.2 Mathematical model of free-surface flow

Ocellaris solves the variable-density Navier–Stokes equations (III.1) to (III.3) with piecewise constant density and viscosity. Standard notation is used for the unknown functions; $\boldsymbol{u}$ is the velocity, p is the pressure, and ρ is the fluid density. The coefficients are the dynamic viscosity μ and the acceleration of gravity g,

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\right) = \nabla\cdot\mu\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right) - \nabla p + \rho\boldsymbol{g}, \tag{III.1}$$

$$\nabla\cdot\boldsymbol{u} = 0, \tag{III.2}$$

$$\frac{\partial\rho}{\partial t} + \boldsymbol{u}\cdot\nabla\rho = 0. \tag{III.3}$$

Ocellaris is currently designed to study air/water free-surface flows at high Reynolds numbers, and does not include a turbulence closure model or the effect of surface tension. The error made by neglecting these effects is small for the included benchmark tests (Kleefsman et al. 2005; Paulsen et al. 2014). The program design is made to be easily extendable and there should be no fundamental problem with including such effects at a later time. For the first

public release, the sole focus of the Ocellaris project has been to show that using higher-order DG methods for free-surface flows is feasible and can be made stable in regard to convective instabilities without compromising mass conservation.

A VOF colour function approach is used for density transport and free-surface capturing (Hirt and Nichols 1981),

$$\rho = \alpha\,\rho_{\text{water}} + (1-\alpha)\,\rho_{\text{air}}, \tag{III.4}$$

where $\alpha \in [0,1]$ is the colour/indicator function which is used as the unknown instead of the fluid density. The kinematic viscosity, $\nu = \mu/\rho$ is computed similarly,

$$\nu = \alpha\,\nu_{\text{water}} + (1-\alpha)\,\nu_{\text{air}}. \tag{III.5}$$

III.3 Discontinuous Galerkin discretisation

The numerical method is an extension of Landet, Mardal, and Mortensen (2020) which builds on Cockburn, Kanschat, and Schötzau (2005) and uses the symmetric interior-penalty (SIP) method for the viscous term (Arnold 1982). This is similar to what is done by Shahbazi, Fischer, and Ethier (2007) and Cockburn, Kanschat, and Schötzau (2007). The domain is discretised as an irregular mesh comprised of tetrahedral cells. Let $\mathcal{T}$ be the set of all cells and $\mathcal{S}$ the set of all facets in the mesh. Polynomial function spaces of degree k on each cell K are denoted $P_k(K)$. These have no continuity at cell boundaries and no inherent boundary conditions.

We use calligraphic typeface to denote operators and sets, bold italic for vectors functions and italic for scalar functions. For nabla the conventions $(\nabla \boldsymbol{u})_{ij} = \partial_j \boldsymbol{u}_i$ and $(\nabla \cdot \boldsymbol{\sigma})_i = \partial_j \sigma_{ij}$ are used. For time derivatives $\boldsymbol{u} = \boldsymbol{u}^{n+1}$ is the unknown trial function and $\boldsymbol{u}^n$ and $\boldsymbol{u}^{n-1}$ are the known values at the previous two time steps. A second-order backwards-differencing formulation, BDF2, is used for time integration. The parameters are $\{\gamma_1, \gamma_2, \gamma_3\} = \{3/2, -2, 1/2\}$, see the first integral in equation (III.14). The convecting velocity $\boldsymbol{w}$ is considered known—which linearises the momentum equation—and $\boldsymbol{w}$ is $\mathrm{H}^{\mathrm{div}}$-conforming such that the flux is continuous, $[\![\, \boldsymbol{w} \,]\!]_{\boldsymbol{n}} = 0$, see section III.4 for details.

The governing equations, (III.1) to (III.3), are cast into the following form: find $\boldsymbol{u} \in [P_2(K)]^3$, $p \in P_1(K)$, and $\alpha \in P_0(K)$ such that

$$\mathcal{A}(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{w}) + \mathcal{B}(p, \boldsymbol{v}) = \mathcal{D}(\boldsymbol{v}) \quad \forall\ \boldsymbol{v} \in [P_2(K)]^3, \tag{III.6}$$

$$\mathcal{C}(\boldsymbol{u}, q) = \mathcal{E}(q) \quad \forall\ q \in P_1(K), \tag{III.7}$$

$$\mathcal{F}(\alpha, \beta; \boldsymbol{w}) = \mathcal{G}(\beta) \quad \forall\ \beta \in P_0(K), \tag{III.8}$$

in the tessellated domain $\mathcal{T}$ subject to

$$\boldsymbol{u} = \boldsymbol{u}_D \quad \text{on Dirichlet boundary facets, } \mathcal{S}_D \subset \mathcal{S} \tag{III.9}$$

$$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}} = \boldsymbol{a} \quad \text{on Neumann boundary facets, } \mathcal{S}_N \subset \mathcal{S} \tag{III.10}$$

The discontinuous Galerkin (DG) method works by breaking integrals over the whole domain into a sum of integrals over each mesh cell $K \in \mathcal{T}$, and defining fluxes of the unknown functions between these cells. The average and jump operators across an internal facet between two cells K^+ and K^- are defined as

$$\{\{u\}\} = \frac{1}{2}(u^+ + u^-), \tag{III.11}$$

$$[\![u]\!] = u^+ - u^-, \tag{III.12}$$

$$[\![\boldsymbol{u}]\!]_{\boldsymbol{n}} = \boldsymbol{u}^+ \cdot \boldsymbol{n}^+ + \boldsymbol{u}^- \cdot \boldsymbol{n}^-. \tag{III.13}$$

where u^+ is the value of u along the internal facet when computed using the shape functions and degrees of freedom related to the K^+ cell and vice versa for u^-. For the exterior facets, which only have one connected cell due to being located on the domain boundary, let the connected cell be denoted K^+ such that $\boldsymbol{n}^+ \cdot [\![\boldsymbol{u}]\!] = \boldsymbol{n}^+ \cdot \boldsymbol{u}^+ = \boldsymbol{n} \cdot \boldsymbol{u}$. Take $\{\{u\}\} = u$ and otherwise let all K^- values related to the non existing element outside the domain be zero.

III.3.1 Momentum equation

The momentum equation (III.1) is discretised using the SIP method for the elliptic term (Arnold 1982) and otherwise using the fluxes from Cockburn, Kanschat, and Schötzau (2005). The application of boundary conditions and the choice of the penalty parameter κ_μ is described in detail in Landet, Mardal, and Mortensen (2020). The flux of pressure is $\hat{p} = \{\{p\}\}$ and the convective flux $\hat{\boldsymbol{u}}^{\boldsymbol{w}}$ is a pure upwind flux. After multiplication with $\boldsymbol{v}$ and integration over $\mathcal{T}$, followed by integration by parts and application of the SIP method to the viscosity, $\mathcal{A}$ can be found as the bilinear part containing $\boldsymbol{u} = \boldsymbol{u}^{n+1}$, $\mathcal{B}$ as the bilinear part containing p, and $\mathcal{D}$ as the linear part of

$$\begin{aligned}
&\int_{\mathcal{T}} \frac{\rho}{\Delta t}(\gamma_1 \boldsymbol{u} + \gamma_2 \boldsymbol{u}^n + \gamma_3 \boldsymbol{u}^{n-1}) \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \\
&\quad - \int_{\mathcal{T}} \boldsymbol{u} \cdot \nabla \cdot (\rho \boldsymbol{v} \otimes \boldsymbol{w}) \,\mathrm{d}\boldsymbol{x} \;+\; \int_{\mathcal{S}} \boldsymbol{w} \cdot \boldsymbol{n}^+ \, \hat{\boldsymbol{u}}^{\boldsymbol{w}} \cdot [\![\rho \boldsymbol{v}]\!] \,\mathrm{d}s \\
&\quad + \int_{\mathcal{T}} \mu \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right) : \nabla \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \;+\; \int_{\mathcal{S}_I} \kappa_\mu [\![\boldsymbol{u}]\!] \cdot [\![\boldsymbol{v}]\!] \,\mathrm{d}s \\
&\quad - \int_{\mathcal{S}} (\{\{ \mu \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right) \}\} \cdot \boldsymbol{n}^+) \cdot [\![\boldsymbol{v}]\!] \,\mathrm{d}s \\
&\quad - \int_{\mathcal{S}_I} (\{\{ \mu \left(\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T \right) \}\} \cdot \boldsymbol{n}^+) \cdot [\![\boldsymbol{u}]\!] \,\mathrm{d}s \\
&\quad - \int_{\mathcal{T}} p \nabla \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \;+\; \int_{\mathcal{S}} \hat{p} \, \boldsymbol{n}^+ \cdot [\![\boldsymbol{v}]\!] \,\mathrm{d}s \;=\; \int_{\mathcal{T}} \rho \boldsymbol{g} \,\mathrm{d}\boldsymbol{x}.
\end{aligned} \tag{III.14}$$

On Dirichlet boundaries $\hat{\boldsymbol{u}}^{\boldsymbol{w}}$ is the upwind value. Depending on flow direction this is either $\boldsymbol{u}$ or $\boldsymbol{u}_D$. On Neumann boundaries let $\hat{\boldsymbol{u}}^{\boldsymbol{w}} = \boldsymbol{u}$. The boundary

conditions (BCs) can be set separately for each velocity component, allowing slip, $\boldsymbol{u} \cdot \boldsymbol{n} = 0$, or non-slip, $\boldsymbol{u} = 0$, BCs. On all boundaries take $\hat{p} = p$.

III.3.2 Continuity equation

The equation used to ensure a divergence-free velocity field (III.2) is multiplied by q and integrated over $\mathcal{T}$. The flux is $\hat{\boldsymbol{u}}^p = \{\{\boldsymbol{u}\}\}$. After integration by parts $\mathcal{C}(\boldsymbol{u}, q)$ and $\mathcal{E}(q)$ can be found from

$$\int_{\mathcal{S}} \hat{\boldsymbol{u}}^p \cdot \boldsymbol{n}^+ [\![q]\!] \, \mathrm{d}s \; - \; \int_{\mathcal{T}} \boldsymbol{u} \cdot \nabla q \, \mathrm{d}\boldsymbol{x} \; = \; 0. \tag{III.15}$$

The non-zero $\mathcal{E}(q)$ results from using the boundary conditions in the flux, $\hat{\boldsymbol{u}}^p = \boldsymbol{u}_D$ on Dirichlet boundaries. On Neumann boundaries the unknown function is used directly, $\hat{\boldsymbol{u}}^p = \boldsymbol{u}$, but when assembling the contribution from each velocity component separately it is beneficial to set $\hat{\boldsymbol{u}}^p = 0$ on free-slip surfaces also for components with Neumann BCs. Otherwise, it is possible to get $\boldsymbol{u} \cdot \boldsymbol{n} \neq 0$ due to small errors in the generated meshes when the boundaries are not perfectly approximated by simplices.

III.3.3 Density transport

The HRIC method (Muzaferija et al. 1998) is used to define the colour function flux $\hat{\alpha}$. The HRIC flux limiter is stable and avoids the excessive interface diffusion of a standard upwind flux. It would be ideal to construct a better interface capturing method that can take advantage of the fact that the velocity is in $[P_2(K)]^3$, but in this first public release of Ocellaris a standard algebraic VOF method is used to validate the stability and applicability of the overall free-surface flow solver.

To construct the DG operators for the density transport equation (III.8), the strong form in equation (III.3) is multiplied by the test function β, and the result is integrated over the domain. After integration by parts and discarding derivatives of the piecewise constant functions, $\mathcal{F}$ can be found as the bilinear part and $\mathcal{G}$ as the linear part of

$$\int_{\mathcal{T}} \frac{1}{\Delta t} (\gamma_1 \alpha^{n+1} + \gamma_2 \alpha^n + \gamma_3 \alpha^{n-1}) \beta \, \mathrm{d}\boldsymbol{x} + \int_{\mathcal{S}} \hat{\alpha}^{n+1} \, \boldsymbol{w} \cdot \boldsymbol{n}^+ [\![\alpha]\!] \, \mathrm{d}s = 0, \tag{III.16}$$

III.3.4 Velocity slope limiter

Using functions spaces that are higher order than piecewise constants to approximate the velocity will cause Gibbs oscillation instabilities when the density field has jumps, such as the order 1000 jump in density in free-surface simulations of air and water. Slope-limiting techniques can be used to remove such instabilities. The velocity slope-limiting approach taken here is the component-wise hierarchical Taylor-polynomial based slope limiter described in Landet, Mardal, and Mortensen (2020), where each velocity component of the convect*ed* velocity,

$\boldsymbol{u}$, is slope limited by a vertex-based scalar slope limiter (Kuzmin 2010) while the convect*ing* velocity, $\boldsymbol{w}$, is left unchanged. This approach requires no tuning and the time spent in the velocity slope limiter is only around 0.2 % of the total running time.

The effect of using a slope limiter is dramatic. Figure III.1 shows the evolution of the total, potential and kinetic energy in the 3D dam-breaking test case presented more thoroughly in section III.7.1. Water rushes out of the broken dam towards a small box-shaped object which is impacted after about 0.4 s. As can be seen, the slope-limited simulation preserves total energy well, trading potential for kinetic energy up to the time of impact. After the impact the conservation of energy is not perfect,but the kinetic energy is always controlled and does not blow up. The second plot shows that the kinetic energy quickly blows up when using the same simulation setup with the velocity slope limiter deactivated. The simulation is automatically stopped after less than 50 time steps when the Courant number passes 1000.

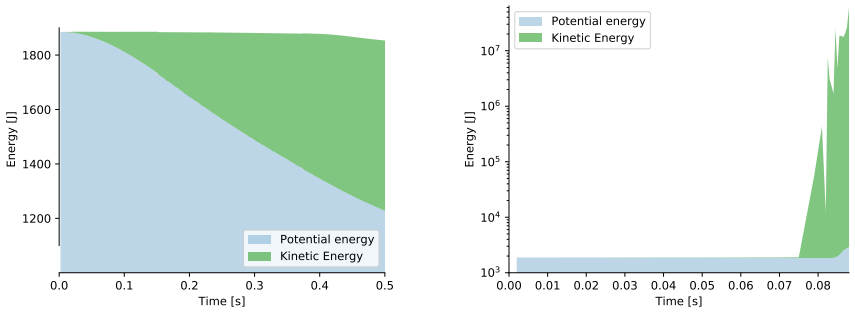


Figure III.1: The evolution of the kinetic, potential and total energy. 3D dam breaking, fine mesh, with (left) and without (right) velocity slope limiting. Notice the difference in time scales between the figures.

III.4 Solution algorithm

The time-stepping procedure is shown in algorithm 1. To decouple the density field solver from the Navier–Stokes solver, and to linearise the momentum equation, a second-order extrapolation $\boldsymbol{w}^*$ is used to estimate the convective velocity $\boldsymbol{w}$,

$$\boldsymbol{w}^* = \boldsymbol{w}^{*,\,n+1} = 2\boldsymbol{w}^n - \boldsymbol{w}^{n-1}, \tag{III.17}$$

which means ρ^{n+1} can be computed before the velocity at time $t = (n+1)\Delta t$.

The discretised matrix version of equations (III.6) and (III.7),

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{d} \\ \mathbf{e} \end{bmatrix}, \tag{III.18}$$

Algorithm 1: The Ocellaris IPCS-A time stepping procedure.

while $t^{n+1} < t_{\max}$ **do**
  Solve for ρ^{n+1} using $\boldsymbol{w}^*$ (estimate of $\boldsymbol{w}^{n+1}$);
  while *not converged* **do**
    Solve momentum equation for $\boldsymbol{u}^*$;
    Solve pressure correction equation for p^{n+1};
    Update the velocity $\boldsymbol{u}^{**}$ using $\boldsymbol{u}^*$ and p^{n+1};
  Compute $\boldsymbol{u}^{n+1}$ by projecting $\boldsymbol{u}^{**}$ into BDM;
  Copy $\boldsymbol{u}^{n+1}$ into $\boldsymbol{w}^{n+1}$;
  Slope limit $\boldsymbol{u}^{n+1}$;
  Increment n;

is a saddle-point problem, which is solved by the incremental pressure-correction scheme on algebraic form (IPCS-A). This is an incremental version of the classic Chorin-Temam methods (Chorin 1968; Temam 1969). $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are sparse matrix versions of the $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ operators, discretised using the discontinuous Galerkin method described in section III.3. The unknowns $\boldsymbol{u}$ and p are now vectors of degrees of freedom, $\mathbf{u}$ and $\mathbf{p}$.

The first IPCS-A step is momentum prediction, performed by using an approximate pressure field $\mathbf{p}^*$ inserted into the first row of equation (III.18),

$$\mathbf{A}\mathbf{u}^* = \mathbf{d} - \mathbf{B}\mathbf{p}^*. \tag{III.19}$$

A splitting error is now introduced. First let $\mathbf{A} = \mathbf{M} + \mathbf{R}$ where $\mathbf{M}$ is a scaled mass matrix resulting from assembly of the time derivative and $\mathbf{R}$ contains the convective and diffusive operators. Then make the assumption that $\mathbf{R}(\mathbf{u} - \mathbf{u}^*) \approx 0$ and use this when subtracting equation (III.19) from the first row of equation (III.18),

$$\mathbf{M}(\mathbf{u} - \mathbf{u}^*) = -\mathbf{B}(\mathbf{p} - \mathbf{p}^*). \tag{III.20}$$

The matrix $\mathbf{M}$ is block diagonal, and is hence cheap to invert. Use this property and the divergence-free criterion, $\mathbf{C}\mathbf{u} = \mathbf{e}$, to remove the unknown $\mathbf{u}$ from equation (III.20), and reorganise this into an equation for $\mathbf{p}$,

$$\mathbf{C}\mathbf{M}^{-1}\mathbf{B}\mathbf{p} = \mathbf{C}\mathbf{M}^{-1}\mathbf{B}\mathbf{p}^* - \mathbf{e} + \mathbf{C}\mathbf{u}^*. \tag{III.21}$$

The velocity $\mathbf{u}$ can be recovered without solving a linear system, simply by substituting the pressure $\mathbf{p}$ from the solution of equation (III.21) into equation (III.20),

$$\mathbf{u} = -\mathbf{M}^{-1}\mathbf{B}(\mathbf{p} - \mathbf{p}^*). \tag{III.22}$$

The momentum-prediction equation (III.19) is solved first, followed by the pressure-correction, equation (III.21) and the velocity update equation (III.22) iteratively until the required accuracy is reached. The resulting velocity field is

projected into a BDM-type function space (Brezzi et al. 1987) where it becomes exactly divergence free—the velocity flux is continuous across internal facets and the velocity field is pointwise divergence free inside each cell, see Landet and Mortensen (2019) for details. The result from this projection is stored both as $\boldsymbol{u}^{n+1}$ and $\boldsymbol{w}^{n+1}$. The last step is slope limiting $\boldsymbol{u}^{n+1}$ such that any instabilities are prevented from growing through the time derivative.

Solving the coupled problem in equation (III.18) without some form of pressure and velocity splitting requires a direct solver, and such solvers scale badly in terms of parallel computing efficiency. The IPCS-A method can be solved using standard iterative Krylov methods which scale much better. Exact mass conservation, $\mathbf{Cu} - \mathbf{e} = 0$, is still ensured on the algebraic level in each iteration,

$$\begin{aligned} \mathbf{Cu} - \mathbf{e} = &\ \mathbf{Cu} - \mathbf{Cu}^* + \mathbf{Cu}^* - \mathbf{e} \\ \overset{\text{(III.20)}}{=} &\ \mathbf{CM}^{-1}\mathbf{B}(\mathbf{p} - \mathbf{p}^*) + \mathbf{Cu}^* - \mathbf{e} \\ \overset{\text{(III.21)}}{=} &\ 0. \end{aligned} \tag{III.23}$$

For the example simulations described in section III.7, the maximum cell wise divergence error in the convecting velocity range from 10^{-5} to 10^{-3}, while the convected velocity (which is slope limited) has a maximum cell wise divergence error ranging from 10^2 to 10^4. This is with 10^{-8} relative error and 10^{-10} absolute error convergence criteria in the pressure-correction Krylov solver. The result is that the simulation in section III.7.1, a dam breaking in a closed box, has less than 3×10^{-5} % change in total mass from time step 3 to the final time step (3415 steps, two seconds simulated time).

III.5 Incoming waves and boundary reflections

In our second numerical example, shown in detail in section III.7.2, we will study a surface-piercing vertical cylinder in an infinite wave field. A truncated computational domain is inevitably required to compute the solution in a finite amount of time. Many methods have been suggested to dampen reflected waves from the artificial computational boundaries, and to quantify the effect of various damping methods, starting with the analytical studies of wave damping by permeable structures in Miche (1944) and Straub, Bowers, and Herbich (1957) and the first papers on similar *sponge layers* in numerical codes (Israeli and Orszag 1981; Larsen and Dancy 1983). Since then, a wealth of methods have appeared.

We have used a forcing-zone approach to wave damping, following the method described in Perić and Abdel-Maksoud (2016) and Perić and Abdel-Maksoud (2018). The only novel aspect is the implementation of the method in a finite element setting, which is straight forward based on the equations presented in the cited papers. We impose Dirichlet boundary conditions on the velocity and density fields at both inlet and outlet boundaries. These boundaries are then padded by forcing zones inside the domain which penalise deviations from the

undisturbed wave field. This damps out any free-surface disturbances caused by the structure without having to damp out the incident wave field itself. See Perić and Abdel-Maksoud (2018) for estimates of the minimum forcing zone size and the penalty magnitude needed to obtain a given reduction in reflected wave amplitudes.

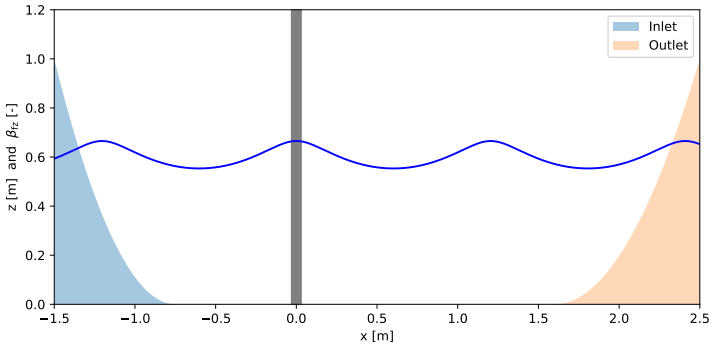


Figure III.2: Forcing zones in the "Cylinder in regular waves" test case.

Figure III.2 shows the forcing zones used in the test case described in section III.7.2. The inlet zone is 0.75 m long and the outlet zone is 1.0 m long. The shape of the zone is a quadratic polynomial with maximum value 1.0 at the boundary and zero first-derivative towards the inner domain. To force the solution towards the incident wave field, a penalty term is added to the momentum equations,

$$\int_{\mathcal{T}} \kappa_{\mathrm{fz}}\, \beta_{\mathrm{fz}} (\boldsymbol{u} - \boldsymbol{u}_D) \cdot \boldsymbol{v}\, \mathrm{d}\boldsymbol{x}, \tag{III.24}$$

where κ_{fz} is the penalty parameter, in our calculations 10, β_{fz} is the zone shape shown in figure III.2 and $\boldsymbol{u}_D(x, z, t)$ is the incident-wave velocity field used for initial and boundary conditions. The same forcing-zone approach is added to the density-transport equation for the colour function with the same forcing-zone shapes and the same penalty parameter, $\kappa_{\mathrm{fz}} = 10$.

The initial and boundary conditions for the velocity and density fields are computed from Fenton stream function wave theory. This is a high-order regular wave theory based on approximating a stream function by a truncated Fourier series. This method of constructing non-linear regular waves was pioneered by Dean (1965). Our implementation is based on Rienecker and Fenton (1981), which is often referred to as "Fenton" stream function wave theory to differentiate it from the original "Dean" stream function wave theory.

The Fenton method is based on collocation in $N+1$ points on the free surface along half the wave length, λ. The Fenton stream function,

$$\Psi_w(x, z, t) = B_0 z + \sum_{j=1}^{N} B_j \frac{\sinh jkz}{\cosh jkD} \cos jkx, \tag{III.25}$$

is non-linear in the wave height η since $z = \eta$ in the collocation points. Ψ_w does a-priori satisfy the bottom boundary condition at $z = 0$ and also the Laplace equation $\nabla^2\Psi_w = 0$. The following conditions are imposed in the Newton–Raphson iterations that are applied to compute the unknown coefficients in equation (III.25): (i) the free surface is a stream line, $\Psi_w = \text{const.}$, (ii) the pressure is constant at the free surface, (iii) the wave height is H, such that $\eta(0) - \eta(\lambda/2) = H$, (iv) the mean wave elevation is D, such that $\int_0^{\lambda/2} \eta \, \mathrm{d}x = D\lambda/2$. See Rienecker and Fenton 1981 for details.

We use the same approach to find a compatible stream function for the air phase, Ψ_a, when Ψ_w has been determined. The stream function in equation (III.25) is now linear in the unknowns since η is known, so the expansion coefficients can be found by a single linear solve. This gives two separate domains where the boundary conditions are satisfied and the divergence is zero, but the velocity parallel to the free surface has a discontinuity at the free surface since the method is based on potential theory and does not contain the viscosity that causes a velocity shear at the free surface. To approximate this, a blended stream function is constructed,

$$\begin{aligned} \Psi &= [1 - f(Z)]\Psi_w(x, z) + f(Z)\Psi_a(x, z), \\ Z &= \frac{z - \eta(x)}{d - \eta(x)}, \end{aligned} \tag{III.26}$$

where the blending function $f(Z)$ is zero in the water and unity in the air above the blending zone. All blending is performed in the air phase, and the blending zone has height d, starting from the free surface. A fifth-order polynomial smooth step function is used for $f(Z)$. This function has zero first and second derivatives at the top and bottom of the blending zone. The resulting velocity field,

$$\begin{aligned} \mathbf{u}_x = \quad & (1 - f)\frac{\partial \Psi_w}{\partial z} + f\frac{\partial \Psi_a}{\partial z} - \frac{df}{dz}\Psi_w(x, z) + \frac{df}{dz}\Psi_a(x, z), \\ \mathbf{u}_z = -& (1 - f)\frac{\partial \Psi_w}{\partial x} - f\frac{\partial \Psi_a}{\partial x} + \frac{df}{dx}\Psi_w(x, z) - \frac{df}{dx}\Psi_a(x, z), \end{aligned} \tag{III.27}$$

is continuous and satisfies the continuity equation everywhere, the dynamic and kinematic free-surface boundary conditions at the interface, and the free-slip boundary conditions at the top and bottom of the domain. These properties make the blended solution a good choice to use as initial and boundary conditions in an exactly divergence-free Navier–Stokes solver.

It should be noted that the blending zone height and shape are ad hoc, though if the (mean) shape of the boundary layer was known it would most likely be possible to approximate it by the above method. We believe the error made in the initial and boundary conditions by this method are much smaller than the more traditional method of applying the wave velocity field in the water phase and setting the velocity in the air to zero, or at most a constant value. By having a continuous stream-function description of the whole field, the initial and boundary conditions are smooth and exactly divergence free pointwise and globally. Enforcing a non-zero-divergence velocity field by Dirichlet boundary

conditions on all boundaries would be incompatible with the incompressibility criterion and make the global system unsolvable. The presented method can be (and is) used with Dirichlet conditions on all normal components without problems.

III.6 Implementation

The Ocellaris solver (Landet 2019c) is built on FEniCS (Logg, Mardal, and Wells 2012) and PETSc (Balay, Abhyankar, et al. 2018; Balay, Gropp, et al. 1997; Dalcin et al. 2011; Davis 2004; LLNL n.d.). The overwhelming majority of the code, including the definition of the weak form and the time-stepping procedure, is implemented in Python 3. The FEniCS form compiler, FFC, is used to compile the weak form, defined in UFL Python format, to optimised C++ code (Alnæs et al. 2014; R. C. Kirby and Logg 2006; Ølgaard, Logg, and Wells 2008). Wherever tight loops over the mesh cells or facets are needed in other parts of the program, the loop is written in C++. For the simulation examples shown in section III.7, the additional C++ code is restricted to parts of the VOF implementation and the velocity slope limiter.

Ocellaris is developed using automated unit and MMS (method of manufactured solutions) testing. Unfortunately, MMS testing of two-phase flows with discontinuous density fields is not well-developed, so testing of this functionality involves running full time simulations, which is not done automatically on each change of the code as this would be too expensive. The Ocellaris program design is made up of independent pluggable components, letting the user define which combination of pressure-splitting scheme, free-surface model, slope limiter and more to use, simply by referencing the relevant components in the input file. This also means that there are automated MMS test of the implemented pressure-correction solvers, but they are tested with a single-phase flow model where classical analytical solutions are available, such as the Taylor–Green vortex (Green and Taylor 1937).

III.6.1 Example input file

The following file listings show excerpts from the input file used to simulate regular waves passing a vertical, surface-piercing cylinder. More information about this example can be found in section III.7.2. Complete documentation of the input file format can be found in (Landet 2019b), and the full input file can be found in (Landet 2019a). All Ocellaris input files are written on YAML format and start with a mandatory header section followed by an optional metadata section. An example of these two input file sections is shown in listing III.1.

```
ocellaris:
  type: input
  version: 1.0
metadata:
  author: Tormod Landet
  date: 2018-12-06
```

```
  description: |
      Surface piercing cylinder with regular waves
```

Listing III.1: Input file header and metadata.

Ocellaris supports defining constants to be used throughout the input file. Defining these once on top of the file makes parameter studies and input changes easier to perform and makes the file easier to read. The definition of convenience constants and physical constants can be seen in listing III.2.

```
user_code:
    constants:
        H: 1.20      # Domain depth
        R: 0.03      # Cylinder radius
        L: 4.00      # Domain length
        B: 0.50      # Domain breadth
        C: 1.50      # Dist. from cylinder (origin) to inlet
        d: 0.60      # Water depth
        w: 0.75      # Length of the forcing zone
        wplus: 0.15  # Additional forcing zone at outlet
physical_properties:
    rho0: 1000.0
    nu0: 1.0e-6
    rho1: 1.0
    nu1: 1.5e-5
    g: [0, 0, -9.81]
```

Listing III.2: Input file constants and physical constants.

The mesh section is shown in listing III.3. The mesh file, created in gmsh (Geuzaine and Remacle 2009), is loaded using the `meshio` Python package which implements readers and writers for many unstructured mesh file formats.

```
  mesh:
      type: meshio
      mesh_file: ../meshA/cylinder.msh
      meshio_type: gmsh
```

Listing III.3: Input file mesh definition.

Known field functions are defined in listings III.4 and III.5. Known fields are used in Ocellaris to define initial and boundary conditions and also to define the location of the free-surface wave damping zones described in section III.5. The waves are defined using the `raschii` Python package which produces C++ code for the Fenton and Stokes regular wave models for use in FEniCS-based solvers. Note also that Ocellaris accepts Python expressions such as `"py$ H - d"` instead of scalars, booleans and strings. These expressions can be used to define parameters in terms of the user-defined constants.

```
fields:
-   name: waves
    type: RaschiiWaves
    wave_model: Fenton
    air_model: FentonAir
    model_order: 10
    still_water_position: py$ d
```

```
    depth: py$ d
    depth_above: py$ H - d
    blending_height: 0.3
    wave_height: 0.11187
    wave_length: 1.20444
```

Listing III.4: Input file definition of the incoming wave field.

The damping zone in listing III.5 is implemented as a generic scalar field. The C++ code is compiled inside a namespace that includes all user-specified constants and an array, `x`, defining the coordinates where the field is to be evaluated. Using C++ lambdas allows using multi-line expressions to compute the field value. Standard C++ expressions, such as `"x[0] + x[1]*x[2]"` $(x + yz)$, can also be given for fields which do not need multiple statements to compute the field value. The inlet damping zone is defined equivalently to the outlet damping zone, but is not shown here for brevity.

```
-   name: outlet zone
    type: ScalarField
    variable_name: beta
    stationary: yes
    cpp_code: |
        [&]() {
            double dz0 = (L - (w + wplus)) - C;
            double dz1 = (L - 0) - C;
            if (x[0] < dz0) {
                return 0.0;
            } else if (x[0] > dz1) {
                return 1.0;
            } else {
                return pow((x[0] - dz0)/(dz1 - dz0), 2);
            }
        }()
```

Listing III.5: Input file definition of the outlet damping zone location.

A definition of a forcing zone is shown in listing III.6. There are four such zones in the simulation, damping the momentum and density fields at the inlet and outlet, see section III.5. The zone in the example shows the damping of the momentum equations at the outlet boundary. It uses the previously defined known fields `waves` and `outlet zone` to specify the target value and field location.

```
forcing_zones:
-   name: outlet velocity damping
    type: MomentumForcing
    zone: outlet zone/beta
    target: waves/u
    penalty: 10
    plot: no
```

Listing III.6: Input file definition of a momentum damping zone.

The initial conditions are defined in listing III.7. The naming scheme uses the postfix `p` to specify the value of a field at $t = 0$, the previous time-step value, and `0` and `2` to specify the x and z directions respectively. To use higher-order

time stepping from the start, the values at $t = -\Delta t$ can be specified by using the `pp` prefix, but that is normally only done in convergence tests where the analytical solution is known.

```
initial_conditions:
    cp:  # c is the VOF colour function
        function: waves/c
    up0:
        function: waves/uhoriz
    up2:
        function: waves/uvert
```

Listing III.7: Input file definition of initial conditions.

Listing III.8 shows the definition of boundary conditions for the inlet. The inside code is used to select the facets on the inlet. If a boundary region is marked with an integer identifier in the mesh generator, then facet selection can be based on this identifier instead. For the cylinder-in-waves example the boundary facets are all marked with C++ code as shown in listing III.8. The `on_boundary` boolean flag is `true` for external facing facets.

```
boundary_conditions:
-   name: Inlet
    selector: code
    inside_code: "on_boundary and x[0] < 0 - C + 1e-5"
    u0:
        type: FieldFunction
        function: waves/uhoriz
    u1:
        type: ConstantValue
        value: 0
    u2:
        type: FieldFunction
        function: waves/uvert
    c:
        type: FieldFunction
        function: waves/c
```

Listing III.8: Input file definition of boundary conditions.

The Navier–Stokes solver section specifies which velocity–pressure splitting scheme to use, the iteration tolerances and the PETSc Krylov solver parameters. Any PETSc configuration variable can be changed for maximum flexibility. In the example shown in listing III.9, the Ocellaris default options for the momentum and pressure PETSc solvers are used and only the convergence criteria and the number of inner iterations (the number of pressure corrections per time step) are changed. After ten time steps, only two pressure corrections are performed per time step. The Krylov solver tolerances are given as three numbers; the tolerance for the first three pressure corrections, the value for the mid range of pressure corrections, and finally the values for the last five pressure corrections. Since, after the first ten time steps, there are only two pressure corrections per time step, only the last value in the lists matter. This gradual decrease of tolerances is done to avoid spending a lot of time in the Krylov solver in the beginning

of the simulation when the pressure corrections are not converged and exact answers are not needed.

```
solver:
  type: IPCS-A
  num_inner_iter: py$ 10 if it < 3 else (5 if it < 10 else 2)
  allowable_error_inner: 1.0e-4
  use_stress_divergence_form: yes
  u:
      inner_iter_control: [3, 5]
      inner_iter_rtol: [1.0e-2, 1.0e-4, 1.0e-6]
      inner_iter_atol: [1.0e-2, 1.0e-4, 1.0e-6]
      inner_iter_max_it: [50, 200, 9999]
  p:
      inner_iter_control: [3, 5]
      inner_iter_rtol: [1.0e-4, 1.0e-6, 1.0e-8]
      inner_iter_atol: [1.0e-6, 1.0e-8, 1.0e-10]
      inner_iter_max_it: [50, 200, 9999]
```

Listing III.9: Input file configuration of the solver.

The multiphase VOF input sections are shown in listing III.10. The HRIC VOF scheme is selected and 5 sub-cycles—the number of advection steps of the VOF colour function per time step of the Navier–Stokes solver—are applied to maximise sharpness of the interface by lowering the effective Courant number in the VOF solver.

```
multiphase_solver:
    type: BlendedAlgebraicVOF
    num_subcycles: 5
convection:
    c:
        convection_scheme: HRIC
```

Listing III.10: Input file configuration of the multi-phase solver.

The final excerpt from the input file is shown in listing III.11. Here the velocity slope limiter is configured to use the scalar `HierarchicalTaylor` slope-limiting method by Kuzmin (2010) for each component of the convected velocity field.

```
slope_limiter:
  u:
      method: Componentwise
      comp_method: HierarchicalTaylor
```

Listing III.11: Input file slope-limiter configuration.

III.7 Numerical examples

The following example simulations have been run in Ocellaris. Source code and input files, which can be used to reproduce all the results, can be found in (Landet 2019a). Visualisations and isosurfaces have been made in Paraview (Ahrens, Geveci, and Law 2005). The meshes, all available for download, have been generated and optimised in Gmsh (Geuzaine and Remacle 2009).

III.7.1 3D dam breaking

Kleefsman et al. (2005) presents experimental and numerical results for a 3D dam-breaking test case where a small container is subjected to a fast-moving front of water that splashes over and around the container. The tank is $3\,\mathrm{m} \times 1\,\mathrm{m} \times 1\,\mathrm{m}$ and the container is $16\,\mathrm{cm} \times 40\,\mathrm{cm} \times 16\,\mathrm{cm}$. There are three surface-height probes that are initially dry and one that is in the fluid (H4). The container is fitted with eight pressure gauges close to the centre line. The locations of the gauges can be found in Issa and Violeau (2006) along with an exact description of the tank and container geometries.

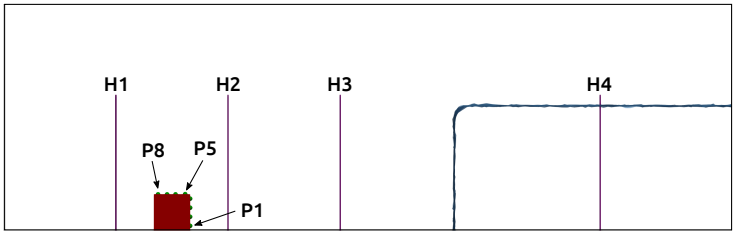


Figure III.3: 3D dam breaking. Sketch of the initial conditions along with the location of the surface-height probes H1–H4 and pressure gauges P1–P8.

The Ocellaris simulations have been run based on irregular tetrahedral meshes. The same mesh input has been used with three different mesh densities, resulting in a coarse mesh with a total of 18 676 tetrahedra, a medium-density mesh with 40 450 tetrahedra and a fine mesh with 123 628 tetrahedra. A cross-sectional view of the medium-density mesh can be seen in figure III.4. The time step is adaptively controlled based on the maximum cell-based (III.28) and facet-based (III.29) Courant numbers. The time step is halved if one of the two is above 0.3 and doubled if they are both below 0.05. The Courant numbers are computed for each cell and facet as

$$\mathrm{Co} = \frac{|\boldsymbol{u}|\Delta t}{D_c} \qquad \text{(cell average)} \tag{III.28}$$

$$\mathrm{Co}_f = \frac{\boldsymbol{u} \cdot \boldsymbol{n}\, S_f \Delta t}{V_c N_{\mathrm{sc}}} \qquad \text{(facet average)} \tag{III.29}$$

where D_c is the cell diameter, S_f is the facet area, and V_c is the cell volume. The facet-based Courant number is used in the HRIC transport scheme for the colour function and this is sub-cycled with N_{sc} sub-cycles per time step. We have used $N_{\mathrm{sc}} = 5$ which makes the cell- and facet-based Courant numbers similar in magnitude.

Figure III.5 shows the evolution of the four free-surface probes for the medium and fine mesh. The fine mesh fits best with the experiments, but both mesh resolutions show good comparison for probes H2–H4. The H1 probe, where the free-surface height is multi valued, does not compare well. It is not clear exactly what heigh is measured in the experiments at this location, we have

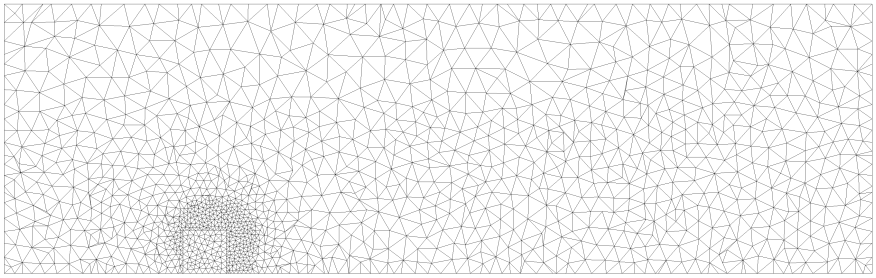


Figure III.4: Longitudinal cut through the centre of the medium-density mesh. The mesh does not conform to the cut plane, making some elements look skewed.

reported the topmost surface intersection. A visualisation of the colour function at $t = 0.4075\,\mathrm{s}$, when pressure probe P2 spikes in the experimental results, is shown in figure III.6. The colour function isosurface from $t = 1.1075\,\mathrm{s}$, when the free surface is multi-valued behind the container, can be seen in figure III.7.

The pressure probe time series are shown in figure III.8. The pressure probes on top of the container, P5–P8, all show very similar behaviour, we have included the results for probe P7 as an example, the other plots would have been roughly identical. On the side of the container facing the wave, the mesh convergence can be seen clearly in the P1 and P2 probes. Further up the container wall, the pressure peak is less pronounced in the numerical results. This is possibly due to the interface being smeared over more cells than optimal, creating a more gradual rise in density at the pressure sensors than what happened in the experiment.

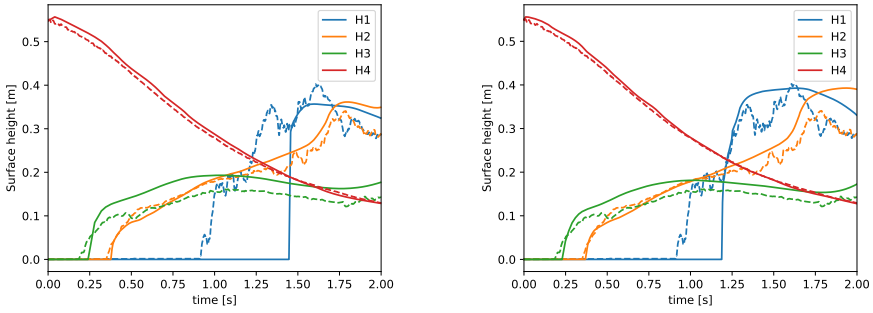


Figure III.5: Free-surface probes. Medium mesh (left) and fine mesh (right). Experimental data shown with dashed lines.

III.7.2 Cylinder in regular waves

Grue and Huseby (2002) gives experimental results for the inline force on a cylinder in regular waves. Figure 3d in that reference is used by Paulsen et

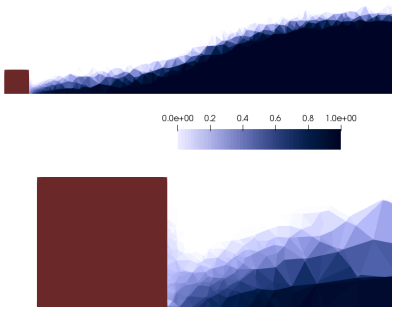


Figure III.6: Fine mesh, $t = 0.4075\,\mathrm{s}$. Centered slice with closeup of the colour function near the container.

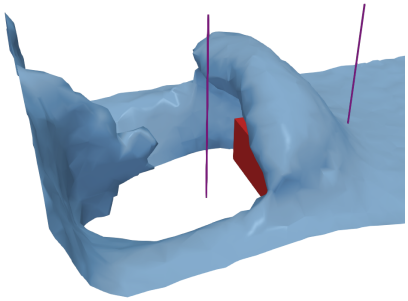


Figure III.7: Fine mesh, $t = 1.1075\,\mathrm{s}$. $\alpha = 0.5$ isosurface from Paraview.

al. (2014) as a benchmark problem, and in this numerical example we will do the same. Fenton stream-function waves are applied in a numerical wave tank which is 4 m long and 0.5 m wide. Our numerical domain height is 1.2 m meter with free-slip BCs at the top. The still-water depth is $h = 0.6\,\mathrm{m}$, the wave heigh is $H = 0.112\,\mathrm{m}$ and the wave length is $\lambda = 1.204\,\mathrm{m}$, leading to a wave number of $k = 5.216\,\mathrm{m}^{-1}$, a wave period of $T = 0.843\,\mathrm{s}$ and a phase speed of $c = 1.429\,\mathrm{m\,s}^{-1}$. The situation is shown in figure III.2 with a surface-piercing vertical cylinder of radius $R = 3\,\mathrm{cm}$. Nondimensional parameters are $kR = 0.16$, $kh = 0.16$ and $kH = 0.58$. The order of the Fenton stream function wave in equation (III.25) is $N = 10$.

The mesh is shown in figure III.9. The mesh cells in the wave propagation zone have characteristic lengths of about 2.5 cm, giving approximately 4.5 elements per wave height. To test wave propagation through the domain a similar mesh is used with the same refinement around the cylinder location, but without the cylinder being present. Symmetry boundary conditions are applied to the centre longitudinal plane shown prominently in figure III.9. The inlet and outlet boundary conditions with associated forcing zones are implemented as described in section III.5. The longitudinal wall away from the cylinder is modelled as free-slip boundary, the same is true for the top and bottom surfaces of the tank.

The cylinder, when it is included, is non-slip in the horizontal plane, but the vertical velocity component is allowed to slip to avoid the free surface sticking to the cylinder. This approximation is done since our mesh resolution is not fine enough to model the true wetting dynamic with appropriate slip lengths. For an introduction to the physics of the contact-line where the free surface intersects the cylinder wall, and how this can be translated into boundary conditions, see e.g. Qian, Wang, and Sheng (2006) and Ren and E (2007). It should be noted that Paulsen et al. (2014) used free-slip boundary conditions in all directions on the cylinder. They were able to get results comparable to the model tests, so the viscous boundary layer does strongly influence the quantities we are comparing.

The difference between the target wave elevation from the Fenton stream

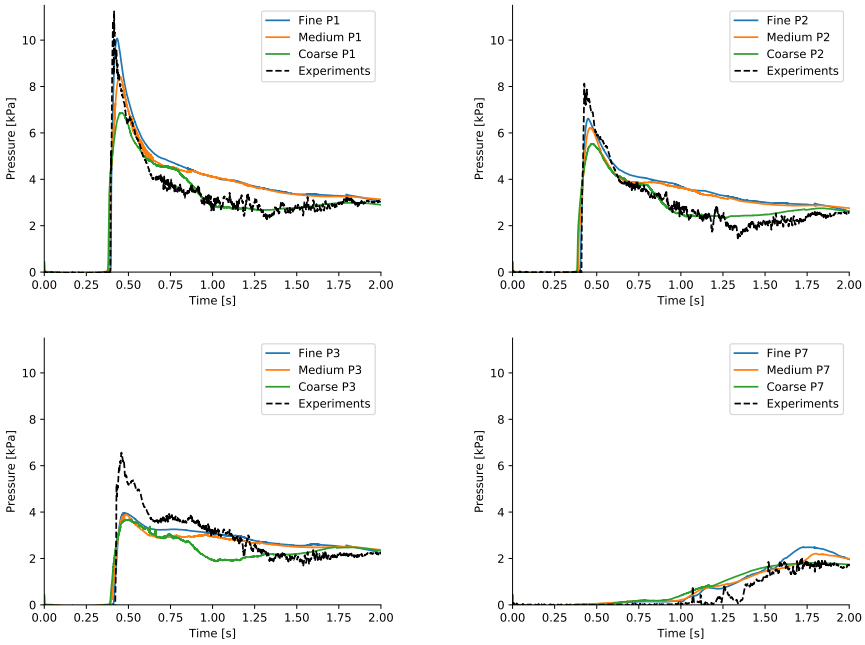


Figure III.8: 3D dam breaking, pressure probe results for all three meshes compared to the time series from the experiments.

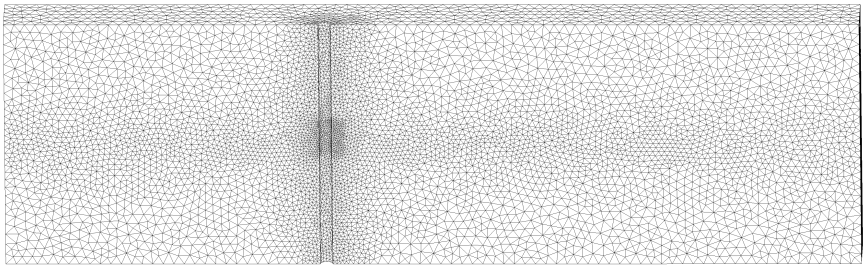


Figure III.9: Mesh used for the cylinder-in-waves test case, only half the domain is included, symmetry is applied on the center face.

function and the VOF free surface is computed based on the intersection of the $\alpha = 0.5$ isosurface and a centred longitudinal plane through the numerical domain without the cylinder. The region from $^2/_3\,\lambda$ in front of the cylinder position to $^1/_3\,\lambda$ behind the cylinder position is used to compute the difference between the phase of the numerical free surface and the stream function's free surface. The diffusive error is computed in the same region as $\frac{1}{H}\sqrt{\langle \eta'^2 \rangle}$ where η' is the difference between the VOF and the stream-function wave elevation after correcting for the phase error and $\langle \cdot \rangle$ denotes the mean over about 300

points where η' is sampled. Figure III.10 shows the results compared to Paulsen et al. (2014, figures 2 and 3), but note that our results are from a 3D domain with forcing zones as described in section III.5, while their results are from 2D simulations in a periodic domain which will let errors develop more easily over time.

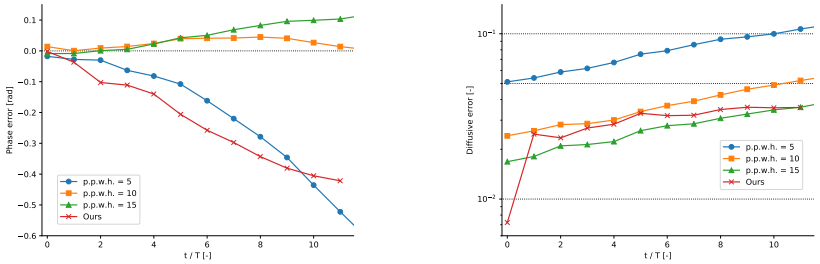


Figure III.10: Phase error (left) and diffusive error (right). Results from Paulsen et al. (2014) where p.p.w.h. means mesh points per wave heigh. Our results have about 4.5 cells per wave height. T is the wave period.

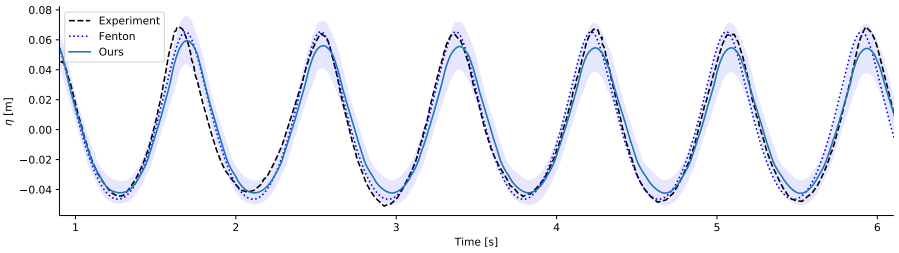


Figure III.11: Surface elevation at the cylinder location, comparing laboratory experiment by Grue and Huseby (2002) to a Fenton stream-function wave and our numerical results. The $\alpha = 0.5$ isosurface is shown along with a shaded area showing $\alpha \in [0.35, 0.65]$.

Our numerical results are compared to the experimental results by Grue and Huseby (2002) in figures III.11 and III.12. The time series from figures 3c and 3d in their work have been shifted in time by $-15.63\,\mathrm{s}$ to align the wave phases. Figure III.11 shows the wave elevation and also includes the Fenton stream-function solution to show the phase and diffusive errors in a more intuitive way. This also confirms that the applied stream-function wave is similar to the wave obtained in the laboratory wave flume. The inline force shown in figure III.12 is computed as the total force on the cylinder in the longitudinal direction, positive towards the outlet. Note the bumps in the force signal right before each trough, the "secondary load cycle". The associated free-surface ridge behind the cylinder at this time is shown in figure III.13.

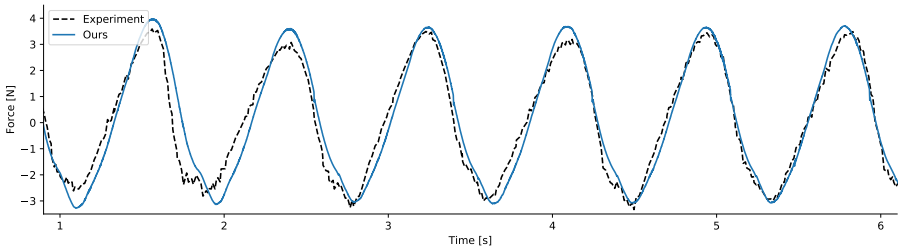


Figure III.12: Inline force on cylinder. Our numerical results compared to the laboratory experiment by Grue and Huseby (2002).

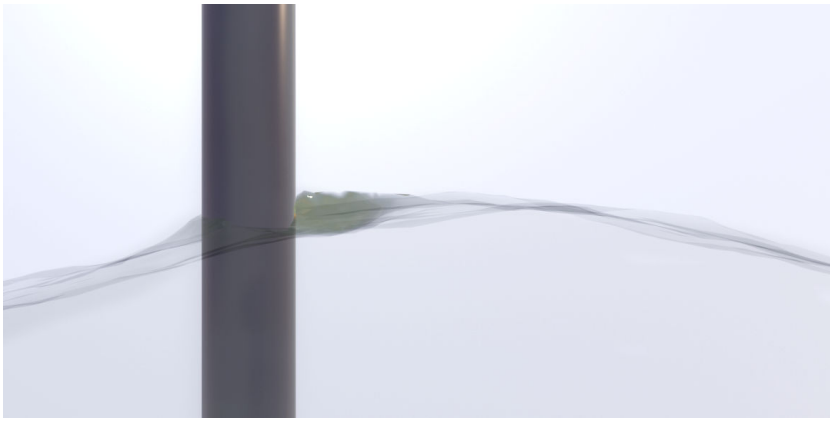


Figure III.13: Snapshot of free surface slightly after a wave crest has passed from left to right. Isosurface made by Paraview and visualised in Blender. The domain was mirrored about the symmetry plane when making the visualisation.

III.8 Discussion

Both test cases presented in section III.7 show that the Ocellaris DG FEM solver is able to simulate complex two-phase flows in 3D. The pressure probes in the dam breaking test show good agreement with the experiments. Pressure probe P1 matches very well and the convergence between the mesh resolutions is clear. The impact pressures higher up on the vertical front wall of the container are less peaked than the measurements. They are most likely softened by the diffusive free surface, which can be seen in figure III.6. The surface height probes match well, except for the probe behind the container where the flow is very complex and the free surface is multi valued. For the second test case the final comparison of the inline force on the cylinder shows an excellent match and the expected free-surface ridge behind the cylinder is observed.

The comparatively primitive free-surface capturing method is the weak point of the current implementation. Using a piecewise constant density field on

the same mesh as the piecewise quadratic convecting velocity means that a lot of the data is thrown away when it comes to the advection of the free surface. Quantities directly related to the jump in the density field will always be limited to first-order convergence, but a more sophisticated free-surface capturing method could provide a sharper profile, hopefully eliminating the softening of the impact pressures in the first test case and removing much of the diffusive error in the wave propagation test. The excellent match of the inline force in the second test case should be treated with some scepticism considering the observed lowering of the peak height of the incident wave. The diffusive interface may have counteracted the expected lowering of the peak force by increasing the air phase pressures through an artificially high fluid density near the free surface.

The reason we selected to use only 4.5 elements per wave height for the wave propagation test is that the velocity function space has 30 unknowns per mesh cell, so it is necessary to limit the number of cells more than in lower-order methods. The simulation has about 120 000 cells and is run on 32 CPUs, so significantly increasing the mesh density will come at a large cost. This highlights the need for more research into free-surface capturing schemes that are designed specifically for high-order methods.

The main advantage of the method is the potential for using fewer elements away from the free surface due to the quadratic approximating polynomials. The dependency on having a very high mesh resolution in the free-surface region must be removed if the method is to be a preferable solution for general free-surface simulations. Another approach would be to make the method h–p adaptive. A relevant issue for further research, which we have not touched on here, is that we observe that the number of iterations in the Krylov solver for the momentum equation increases a lot when the density ratio increases. The momentum equation can most likely be preconditioned or stabilised to stop this tendency.

We have shown that it is viable to use Ocellaris' exactly divergence-free high-order discontinuous Galerkin finite element method with velocity slope limiting to simulate realistic air/water free-surface physics, but some work remains to be fully competitive with established low-order methods.

Acknowledgements

The simulations were performed on resources provided by UNINETT Sigma2, the national infrastructure for high performance computing and data storage in Norway.

References

Ahrens, J., Geveci, B., and Law, C. (2005). "ParaView: An End-User Tool for Large-Data Visualization". In: *Visualization Handbook*. Ed. by Hansen, C. D. and Johnson, C. R. Burlington: Butterworth-Heinemann, pp. 717–731.

Alnæs, M. S. et al. (Mar. 2014). "Unified Form Language: A Domain-specific Language for Weak Formulations of Partial Differential Equations". *ACM Trans. Math. Softw.* 40.2, 9:1–9:37.

Arnold, D. N. (1982). "An interior penalty finite element method with discontinuous elements". *SIAM journal on numerical analysis* 19.4, pp. 742–760.

Babuška, I. and Dorr, M. R. (1981). "Error estimates for the combined h and p versions of the finite element method". *Numerische Mathematik* 37.2, pp. 257–277.

Balay, S., Abhyankar, S., et al. (2018). *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.9. Argonne National Laboratory.

Balay, S., Gropp, W. D., et al. (1997). "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries". In: *Modern Software Tools in Scientific Computing*. Ed. by Arge, E., Bruaset, A. M., and Langtangen, H. P. Birkhäuser Press, pp. 163–202.

Brezzi, F. et al. (1987). "Mixed finite elements for second order elliptic problems in three variables". *Numerische Mathematik* 51.2, pp. 237–250.

Chorin, A. J. (1968). "Numerical solution of the Navier-Stokes equations". *Mathematics of computation* 22.104, pp. 745–762.

Cockburn, B., Kanschat, G., and Schötzau, D. (2005). "A locally conservative LDG method for the incompressible Navier-Stokes equations". *Mathematics of Computation* 74.251, pp. 1067–1096.

— (2007). "A Note on Discontinuous Galerkin Divergence-free Solutions of the Navier–Stokes Equations". *Journal of Scientific Computing* 31.1-2, pp. 61–73.

Dalcin, L. D. et al. (2011). "Parallel distributed computing using Python". *Advances in Water Resources* 34.9. New Computational Methods and Software Tools, pp. 1124–1139.

Davis, T. A. (2004). "Algorithm 832: UMFPACK V4.3—An Unsymmetric-Pattern Multifrontal Method". *ACM Transactions on Mathematical Software (TOMS)* 30.2, pp. 196–199.

Dean, R. G. (1965). "Stream function representation of nonlinear ocean waves". *Journal of Geophysical Research* 70.18, pp. 4561–4572.

Geuzaine, C. and Remacle, J.-F. (2009). "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". *International Journal for Numerical Methods in Engineering* 79.11, pp. 1309–1331.

Green, A. and Taylor, G. (1937). "Mechanism of the production of small eddies from larger ones". In: *Proceedings of the royal society of London. Series A, mathematical and physical sciences*. Vol. 158, pp. 499–521.

Grue, J. and Huseby, M. (2002). "Higher-harmonic wave forces and ringing of vertical cylinders". *Applied Ocean Research* 24.4, pp. 203–214.

Harten, A. (1983). "High resolution schemes for hyperbolic conservation laws". *Journal of Computational Physics* 49.3, pp. 357–393.

Hirt, C. W. and Nichols, B. D. (1981). "Volume of fluid (VOF) method for the dynamics of free boundaries". *Journal of Computational Physics* 39.1, pp. 201–225.

Huerta, A. et al. (2013). "Efficiency of high-order elements for continuous and discontinuous Galerkin methods". *International Journal for Numerical Methods in Engineering* 96.9, pp. 529–560.

Israeli, M. and Orszag, S. A. (1981). "Approximation of radiation boundary conditions". *Journal of Computational Physics* 41.1, pp. 115–135.

Issa, R. and Violeau, D. (2006). *ERCOFTAC Test-case 2, 3D Dambreaking, Release 1.1*. SPH European Research Interest Community SIG, Electricite De France, Laboratoire National Hydaulique et Environnement.

Kirby, R. C. and Logg, A. (Sept. 2006). "A compiler for variational forms". *ACM Transactions on Mathematical Software* 32.3, pp. 417–444.

Kirby, R. M., Sherwin, S. J., and Cockburn, B. (2012). "To CG or to HDG: A Comparative Study". *Journal of Scientific Computing* 51.1, pp. 183–212.

Kleefsman, K. M. T. et al. (2005). "A Volume-of-Fluid based simulation method for wave impact problems". *Journal of Computational Physics* 206.1, pp. 363–393.

Krivodonova, L. et al. (2004). "Shock detection and limiting with discontinuous Galerkin methods for hyperbolic conservation laws". *Applied Numerical Mathematics*. Workshop on Innovative Time Integrators for PDEs 48.3, pp. 323–338.

Kubatko, E. J. et al. (2009). "A Performance Comparison of Continuous and Discontinuous Finite Element Shallow Water Models". *Journal of Scientific Computing* 40.1, pp. 315–339.

Kuzmin, D. (Apr. 2010). "A vertex-based hierarchical slope limiter for p-adaptive discontinuous Galerkin methods". *Journal of Computational and Applied Mathematics*. Finite Element Methods in Engineering and Science (FEMTEC 2009) 233.12, pp. 3077–3085.

Landet, T. (Jan. 2019a). *Ocellaris DG-FEM software and input files to reproduce results*. Zenodo: 10.5281/zenodo.2587038.

— (2019b). *Ocellaris web page and user manual*. www.ocellaris.org.

— (2019c). "Ocellaris: a DG FEM solver for free-surface flows". *The Journal of Open Source Software* 4.35, p. 1239.

Landet, T., Mardal, K.-A., and Mortensen, M. (2020). "Slope limiting the velocity field in a discontinuous Galerkin divergence-free two-phase flow solver". *Computers & Fluids* 196, p. 104322.

Landet, T. and Mortensen, M. (2019). "On exactly incompressible DG FEM pressure splitting schemes for the Navier-Stokes equation". *arXiv:1903.11943 [physics]*.

Larsen, J. and Dancy, H. (1983). "Open boundaries in short wave simulations — A new approach". *Coastal Engineering* 7.3, pp. 285–297.

LLNL (n.d.). *hypre: High Performance Preconditioners*. www.llnl.gov/CASC/hypre/. Lawrence Livermore National Laboratory.

Logg, A., Mardal, K.-A., and Wells, G. (Feb. 2012). *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Springer Science & Business Media.

Miche, M. (1944). "Mouvements ondulatoires de la mer en profondeur constante ou décroissante". *Annales de Ponts et Chaussées*.

Muzaferija, S. et al. (1998). “A Two-Fluid Navier-Stokes Solver to Simulate Water Entry”. In: *Proceedings from the 22nd symposium on naval hydrodynamics*. Washington, DC, pp. 277–289.

Ølgaard, K., Logg, A., and Wells, G. (Nov. 2008). “Automated Code Generation for Discontinuous Galerkin Methods”. *SIAM Journal on Scientific Computing* 31.2, pp. 849–864.

Paulsen, B. T. et al. (2014). “Forcing of a bottom-mounted circular cylinder by steep regular water waves at finite depth”. *Journal of Fluid Mechanics* 755, pp. 1–34.

Perić, R. and Abdel-Maksoud, M. (Jan. 2016). “Reliable damping of free-surface waves in numerical simulations”. *Ship Technology Research* 63.1, pp. 1–13.

Perić, R. and Abdel-Maksoud, M. (2018). “Analytical prediction of reflection coefficients for wave absorbing layers in flow simulations of regular free-surface waves”. *Ocean Engineering* 147, pp. 132–147.

Popinet, S. (2003). “Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries”. *Journal of Computational Physics* 190.2, pp. 572–600.

Qian, T., Wang, X.-P., and Sheng, P. (2006). “Molecular hydrodynamics of the moving contact line in two-phase immiscible flows”. *Communications in Computational Physics* 1.1, pp. 1–52.

Ren, W. and E, W. (2007). “Boundary conditions for the moving contact line problem”. *Physics of Fluids* 19.2, p. 022101.

Rienecker, M. M. and Fenton, J. D. (1981). “A Fourier approximation method for steady water waves”. *Journal of Fluid Mechanics* 104, pp. 119–137.

Shahbazi, K., Fischer, P. F., and Ethier, C. R. (2007). “A high-order discontinuous Galerkin method for the unsteady incompressible Navier–Stokes equations”. *Journal of Computational Physics* 222.1, pp. 391–407.

Straub, L. G., Bowers, C. E., and Herbich, J. B. (1957). “Laboratory tests of permeable wave absorbers”. *Coastal Engineering Proceedings* 1.6, p. 44.

Temam, R. (1969). “Sur l’approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires (II)”. *Archive for rational mechanics and analysis* 33.5, pp. 377–385.

Ubbink, O. (1997). “Numerical prediction of two fluid systems with sharp interfaces”. PhD thesis. Imperial College, University of London.

Weller, H. G. et al. (1998). “A tensorial approach to computational continuum mechanics using object-oriented techniques”. *Computers in Physics* 12.6, pp. 620–631.

IV

Paper IV

Ocellaris: a DG FEM solver for free-surface flows

Tormod Landet

Published in Journal of Open Source Software, DOI: 10.21105/joss.01239

Free-surface flows are found wherever two immiscible fluids come into contact, such as at the interface between water and air in the ocean. Simulations of free-surface flows at high Reynolds numbers are important for the design of coastal, bottom-fixed, and floating structures exposed to ocean waves, as well as partially filled pipes and tanks. The large difference in density between water and air poses problems for numerical approximation across the interface, and the highly non-linear behaviour of the free surface, which can break and overturn, makes separating computations into two different fluid domains difficult. As a model for free-surface flows, Ocellaris solves the variable-density incompressible Navier–Stokes equations with discontinuous density fields,

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\right) = \nabla\cdot\mu\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right) - \nabla p + \rho\boldsymbol{g},$$
$$\nabla\cdot\boldsymbol{u} = 0,$$
$$\frac{\partial \rho}{\partial t} + \boldsymbol{u}\cdot\nabla\rho = 0.$$

Low-order finite volume methods (FVM) are currently the most popular methods for solving the above equations when simulating free-surface flows at high Reynolds numbers[1]. Open source FVM codes include OpenFOAM (Weller et al. 1998) and Gerris (Popinet 2003). Potential flow methods are also used for simulation of non-breaking ocean waves (Tong et al. 2019), and ship-wave interaction (Faltinsen 2005; Kring et al. 1997), but these methods require that viscous effects and vorticity can be disregarded. Finite volume methods are able to include these effects, and can produce exactly incompressible velocity fields. By this we mean that the velocity is pointwise divergence-free, the velocity facet fluxes sum to zero for each cell, and the velocity facet flux is continuous between neighbouring cells. In the mentioned FVM programs, the free-surface advection is implemented with the volume-of-fluid (VOF) method (Hirt and Nichols 1981), which requires that the advected fluid-indicator function c is bounded, $c \in [0, 1]$.

[1] FINE/Marine, FLOW-3D, Fluent, Orca3D, SHIPFLOW XCHAP, and StarCCM+ are examples of proprietary FVM free-surface flow solvers used in the industry.

Numerical VOF advection methods rely upon divergence-free velocity fields to ensure mass conservation and bounded transport.

Due to the piecewise constant discretisation, low-order FVM methods do not suffer from Gibbs oscillations near the large jump in density and momentum at the interface between the immiscible fluids, as long as an appropriate flux limiter is applied (Leonard 1979). But there are some downsides to using low-order methods. Increasing the approximation order is more computationally efficient than increasing the mesh density in areas where the solution is expected to be smooth, which for wave simulations is most of the domain away from the free surface. Implementing higher-order FVM methods is complicated on unstructured meshes, due to the need for large reconstruction stencils in order to obtain higher-order approximations. Higher-order finite element methods (FEM), such as the discontinuous Galerkin method (DG FEM), uses higher-order basis functions in each cell to overcome this problem. The discontinuous nature of DG FEM methods additionally allows more computation to be performed locally with less coupling of cells, which can be beneficial for the overall computational efficiency (see e.g. Kirby, Sherwin, and Cockburn (2012) and Kubatko et al. (2009)).

Ocellaris is an exactly incompressible Navier–Stokes solver for free-surface flows with a DG FEM based numerical method that supports higher-order finite elements and contains specially designed slope-limiting stabilisation filters to be able to handle large density transitions (Landet 2019; Landet, Mardal, and Mortensen 2018). Ocellaris is implemented in Python and C++ with FEniCS (Alnæs et al. 2015; Logg, Mardal, Wells, et al. 2012) as the backend for the mesh and finite element assembly. PETSc is used for solving the resulting linear systems (Balay, Abhyankar, et al. 2018; Balay, Gropp, et al. 1997; Dalcin et al. 2011; Davis 2004; *hypre: High Performance Preconditioners* n.d.).

Ocellaris uses a YAML based input file format documented in the user guide available at ocellaris.org. The mesh geometry can be defined directly in the input file for simple geometries, or it can be loaded from any file format supported by meshio (Schlömer 2019), as long as the unstructured meshes are simplicial; triangles in 2D and tetrahedra in 3D. Due to the flexible nature of the implementation, custom numerical models can be added to a simulation by referencing external Python modules from the input file. Ocellaris uses the XDMF file format (*The XDMF format* 2019) for visualisations and a custom HDF5 format for restart files (*The HDF5 format* 2019).

References

Alnæs, M. S. et al. (2015). "The FEniCS Project Version 1.5". *Archive of Numerical Software* 3.100.

Balay, S., Abhyankar, S., et al. (2018). *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.9. Argonne National Laboratory.

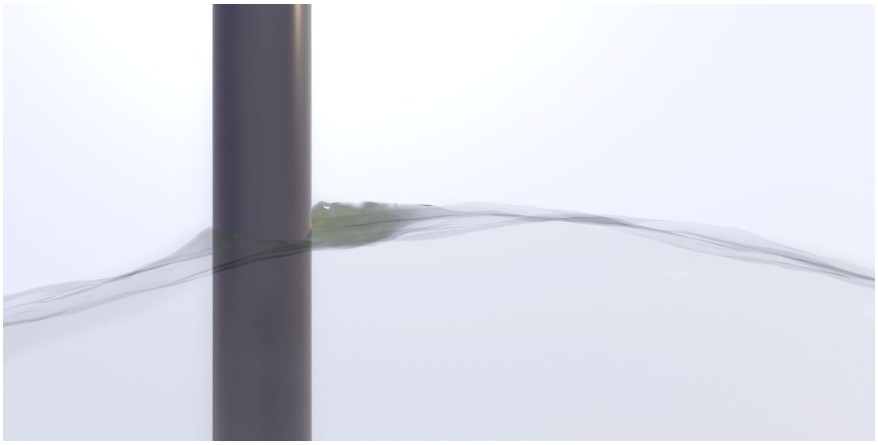


Figure IV.1: Cylinder in waves. Rendered in Blender after using Paraview to extract the free surface from an Ocellaris simulation.

Balay, S., Gropp, W. D., et al. (1997). "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries". In: *Modern Software Tools in Scientific Computing*. Ed. by Arge, E., Bruaset, A. M., and Langtangen, H. P. Birkhäuser Press, pp. 163–202.

Dalcin, L. D. et al. (2011). "Parallel distributed computing using Python". *Advances in Water Resources* 34.9. New Computational Methods and Software Tools, pp. 1124–1139.

Davis, T. A. (2004). "Algorithm 832: UMFPACK V4.3—An Unsymmetric-Pattern Multifrontal Method". *ACM Transactions on Mathematical Software (TOMS)* 30.2, pp. 196–199.

Faltinsen, O. M. (2005). *Hydrodynamics of High-Speed Marine Vehicles*. Cambridge University Press.

Hirt, C. W. and Nichols, B. D. (1981). "Volume of fluid (VOF) method for the dynamics of free boundaries". *Journal of Computational Physics* 39.1, pp. 201–225.

hypre: High Performance Preconditioners (n.d.). www.llnl.gov/CASC/hypre/. Lawrence Livermore National Laboratory.

Kirby, R. M., Sherwin, S. J., and Cockburn, B. (2012). "To CG or to HDG: A Comparative Study". *Journal of Scientific Computing* 51.1, pp. 183–212.

Kring, D. et al. (1997). "Nonlinear Ship Motions and Wave-Induced Loads by a Rankine Method". In: *Twenty-First Symposium on Naval Hydrodynamics* (June 24–28, 1996). Trondheim, Norway: The National Academies Press, Washington, DC, pp. 45–63.

Kubatko, E. J. et al. (2009). "A Performance Comparison of Continuous and Discontinuous Finite Element Shallow Water Models". *Journal of Scientific Computing* 40.1, pp. 315–339.

Landet, T. (2019). *The Ocellaris project web page, user guide and blog*. www.ocellaris.org.

Landet, T., Mardal, K.-A., and Mortensen, M. (2018). "Slope limiting the velocity field in a discontinuous Galerkin divergence free two-phase flow solver". *arXiv:1803.06976 [physics]*. arXiv: 1803.06976.

Leonard, B. P. (July 1979). "Adjusted quadratic upstream algorithms for transient incompressible convection". In: *4th Computational Fluid Dynamics Conference*. Williamsburg, VA, U.S.A: American Institute of Aeronautics and Astronautics, pp. 226–233.

Logg, A., Mardal, K.-A., Wells, G. N., et al. (2012). *Automated Solution of Differential Equations by the Finite Element Method*. Springer.

Popinet, S. (2003). "Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries". *Journal of Computational Physics* 190.2, pp. 572–600.

Schlömer, N. (2019). *meshio*. github.com/nschloe/meshio.

The HDF5 format (2019). www.hdfgroup.org.

The XDMF format (2019). xdmf.org.

Tong, C. et al. (May 2019). "Numerical analysis on the generation, propagation and interaction of solitary waves by a Harmonic Polynomial Cell Method". *Wave Motion* 88, pp. 34–56.

Weller, H. G. et al. (1998). "A tensorial approach to computational continuum mechanics using object-oriented techniques". *Computers in Physics* 12.6, pp. 620–631.

Back matter

Concluding words

Conclusion

A novel higher-order method for simulation of free surface flows has been presented. The new method adds a slope limiting strategy for solenoidal velocity fields to an existing single-phase DG FEM Navier–Stokes solution method. Both the selected DG SIP method for solving the incompressible Navier–Stokes equations and the new slope limiter are exactly mass conserving. This means that at any point inside an element the divergence of the (convecting) velocity field is machine-precision zero, and that the normal flux across element boundaries is continuous. The resulting solver is shown to be able to handle the factor-1000 jump in the density field between simulated water and air phases, and both 2D and 3D simulations compare favourably to model tests.

Novelty

In the search for more efficient and accurate methods for free-surface flows, it is important to start exploring the use of higher-order numerical methods. The main contribution of this thesis is using an exactly divergence-free higher-order discontinuous Galerkin finite element methods (DG FEM) for solving the variable-density Navier–Stokes equations in the presence of the large jump in density found in air/water flows.

The work consists of four main parts. Paper I presents a new slope-limiting stabilisation method for velocity fields, and forms the main building block of the thesis. The new method allows the convected velocity field to be slope limited, and hence stabilised, while the convecting velocity field remains exactly divergence free. The results from the new method are compared to free-surface model tests (2D), such as the classic dam-break test.

Paper II compares existing pressure-splitting methods for solving the Navier–Stokes equations with a focus on fast solutions of the DG SIP discretisation used in the first paper. Special attention is put on maintaining exact incompressibility, and the paper discusses the specific choices that must be taken in order to maintain machine-zero divergence. The paper makes a recommendation for which method to use.

Paper III shows 3D simulations using the slope limiter from the first paper coupled with the best-performing pressure-splitting scheme from the second paper. Paper III additionally features the inclusion of an existing forcing-zone approach to handle traveling free-surface waves in a finite domain without reflection. This is used with an existing potential-theory model for regular waves. In order to be compatible, and hence provide exactly divergence-free initial and

boundary conditions, the potential-theory wave description is almost adequate, but a novel blending method is included as a simple solution to the problem posed by the discontinuity in the (potential theory) velocity field at the free surface.

The fourth part of the work is the fully documented numerical solver code, written in the FEniCS framework for solving PDEs, and briefly described in Paper IV. The common theme throughout the work is the divergence-free velocity constraint and the use of higher-order methods without smoothing of the coefficient fields.

Suggestions for further work

I hope that having a relatively simple, numerically stable, well documented, and exactly mass-conserving higher-order method to build upon will make it easier and more attractive to explore numerical methods for the study of free-surface physics with higher-order methods. It is with this hope in mind that the full code and input files to all the presented simulations are made available. Some possible next steps for such research work are outlined below.

Free surface capturing

The two-phase free surface simulations presented in this work have been performed with a standard piecewise-constant Volume-of-Fluid (VOF) method. A relatively simple extension would be to use a refined mesh for the VOF calculations, dividing each cell used in the higher-order Navier–Stokes solver into multiple sub-cells used for the low-order VOF solver. Currently the implemented HRIC VOF solver uses a very small fraction of the computational resources compared to the Navier–Stokes solver. The inability of the method to resolve sub-cell features of the free surface forces the use of meshes that are finer than what would potentially be needed if sub-cell features could be resolved. Hence, using a courser mesh resolution for the pressure and velocity, and finer resolution for the VOF colour function, could lead to a more efficient method where the VOF calculations use a larger fraction of the computational resources, while giving a similar or better resolved solution at lower overall computational cost.

The density field is either constant or has a sharp jump in each cell, so it is not given that using a higher-order method will give any benefits. That being said, there are multiple new methods for surface capturing and level-set/VOF coupling available in recent publications, see the discussion in sections 1.3 and 1.4 for an overview. Comparing such methods to a low-order mass-conserving VOF method on a sub-divided mesh would be an interesting topic for a paper.

Extended FEM

Just like the sub-cell capturing of the free surface discussed above, allowing sub-cell discontinuities in the momentum to be resolved by augmenting the

approximating functions in the intersected cells with a sharp jump at the free surface could give large benefits such as reduced need for slope limiting and hence higher order solutions close to the free surface. XFEM methods that do this have been explored in the literature, e.g. to resolve the jump in pressure at the free surface when surface tension is included (Groß and Reusken 2007, see *Free surface description* in section 1.3). It is not given that including a jump in the (continuous) velocity will stabilise the jump in the (discontinuous) momentum. Neither is it necessarily straight forward to find the exact position of the free surface without using some form of geometric reconstruction instead of the algebraic VOF method used here.

Matrix-free multi-grid methods

Recent research on fast scalable solvers for higher-order DG methods shows that the solution process can be made significantly faster by combining specially tailored multigrid pre-conditioners with matrix-free solvers (see e.g. Fehn, Munch, Wall, and Kronbichler's preprint from 2019 along with their prior work listed in the literature review in section 1.4). The specialised multi-grid preconditioner converts from discontinuous to continuous representations as a part of the multi-grid cycle, and is hence less affected by the magnitude of the penalty factor required for coercivity of elliptic terms. By combining this pre-conditioner with a matrix-free solver, the solution of standard DG SIP formulations (as used in this work) can be made much faster, and even faster than equivalent methods for hybridised DG formulations, which is the other obvious way to increase the speed of the DG Navier–Stokes solver. While algorithmic improvements, such as adaptivity (discussed below), are very important, fast linear-algebra solvers for the existing method could also be a very interesting path for future development.

Adaptivity

As mentioned in the introduction, the most efficient numerical method is h-adaptive close to discontinuities and P-adaptive in areas where the solution is smooth—it uses small low-order elements to resolve any discontinuities and larger high-order elements in smooth areas to optimise resolving power per degree of freedom. Free-surface flows are characterised by the presence of a discontinuity in the density field, so including some form of h-adaptivity would be a good starting point along with support for varying the polynomial order in the domain. This task is not small, and a successful implementation must not only support dynamic remeshing in a stable and controllable way, it must also do this in a very efficient manner such that computational cost is significantly lower than in a non-adaptive code to justify the added complexity. There exist codes that are able to do this (see e.g Basilisk referenced in section 1.2), but these codes are still works in progress and are low order only. It is likely that an efficient h-adaptive method must be written from the ground up for fast mesh refinement to really reap the full benefits of adaptivity.

Consistency of viscous term

The standard symmetric interior penalty method for dealing with the elliptic viscous term in discontinuous Galerkin discretisations will not work if a slope limiter flattens all derivatives, unlike a finite-volume method which reconstructs the viscous fluxes from the cell average values. For areas of the solution where the viscosity is important, such errors in the viscous flux can lead to inconsistencies. It is possible to construct DG methods for elliptic terms that are equivalent to cell-centered finite-volume discretisations in the piecewise-constant case. The studies shown in this work have all been convection dominated, but it would be an interesting study to see the influence of this inconsistency by implementing a consistent discretisation of the elliptic term and compare with the existing standard DG SIP implementation.

Initial and boundary conditions for wave simulations

In Paper III a blending function is introduced ad-hoc to provide a divergence free and continuous velocity field for the Fenton stream-function waves and the air-phase above. This blending function determines the initial boundary layer position, shape and height. By investigating waves in the laboratory with velocimetry to capture the real air/water wave boundary layer, performing numerical simulations, or obtaining such velocity fields from published literature, it could be possible to provide an analytical velocity field that more closely approximates the combined air and water flow near a free-surface wave. By basing the method on potential theory for the physics of the two domains it will of course not be in absolute agreement with the real physics, but this approach enables analytical expressions and—since viscosity has a limited influence away from the free surface—it could be possible to get a good approximation by localised stream-function blending similar to what is described in the Paper III. This would enable enforcing initial and boundary conditions which are closer to the real Navier–Stokes solution of the problem, and a perhaps such an analytical combined velocity field could be used as a simplified tool for the study of air/wave interaction.

Slope limiters

As discussed in the introduction (and explored further in the appendix), using the hierarchical slope limiter close to the boundaries can lead to excessive limiting due to the lack of known "safe" bounds on the limited function and its derivatives on the out-of-domain side of the boundary. This can be remedied by using an immersed boundary formulation, implementing ghost cells layers or using a slope limiter close to the wall that subdivides the mesh into a finer sub-mesh and uses a low-order stable method as a slope limiter. The last option requires re-solving the problem locally using a different method and may hence require the solver to be implemented twice, in addition to sub-mesh generation and projections to and from the low order method, so it may be a quite large undertaking.

It would also be very interesting to compare the slope limiter to e.g. a spectral filter for regular domains such as numerical wave flume. As the slope limiter is a purely explicit post-processing operator there is quite some room to be creative as the coupling to the rest of the code is low. It would also be a nice addition to compare the current component-wise slope limiter to a limiting procedure working on the velocity magnitude and direction fields. This would highlight any influence of the coordinate-system dependency of the current slope limiter implementation.

Appendices

Appendix A

Example: Lid-driven cavity flow

A.1 Introduction

The lid-driven cavity flow simulation is a classic benchmark for 2D flow where the domain consists of a unit square with rigid walls on three sides and an enforced horizontal velocity on the top face, the moving lid, see figure A.1. Here we take one of the parameter choices from the reference solutions in U. Ghia, K. N. Ghia, and Shin (1982). A kinematic viscosity of $\nu = 0.001$ is used, leading to a Reynolds number of 1000. The result, after the simulation has been given some time to stabilise, is a steady-state vortex flow in the domain. The velocity profiles in the horizontal and vertical directions are studied by taking a vertical and a horizontal cut through the centre of the domain and recording the steady-state velocity components.

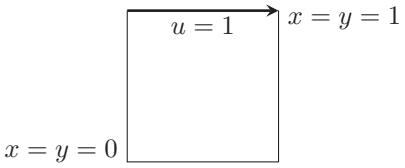


Figure A.1: The driven cavity benchmark geometry.

Ocellaris can read files generated by external meshing programs, but to generate a suitable mesh for the driven cavity problem—with refinement towards the top corners in the square domain—it is perhaps easiest to generate a perfectly regular mesh by the built-in mesh generation routines and then tell Ocellaris to move the mesh according to a given function. Here this option is used and a displacement-function is specified to move the x-coordinates of the mesh nodes towards the vertical walls and the y-coordinates towards the lid. Given a mesh of a domain with size 1.0 m by 1.0 m with the origin in the lower left corner, the displacement is such that the change is -0.07 m at $x = 0.1$ m and 0.07 m at $x = 0.9$ m. A polynomial is fitted to these criteria in addition to the requirement that the displacement of the walls is zero. Similarly, the mesh nodes are moved vertically 0.07 m at $y = 0.9$ m. The nodal displacement function is then

$$x_{\text{new}} = x - 1.9444x^3 + 2.9167x^2 - 0.9723x, \tag{A.1}$$

$$y_{\text{new}} = y - 0.7778y^2 + 0.7778y. \tag{A.2}$$

A.2 Results

The simulation described above was run twice, the only difference being whether the slope limiter was turned on or not. The starting mesh was 40×40 squares, divided into two triangles each. The IPCS-A pressure-correction solver was used with two inner iterations per time step. The results can be seen in figure A.2. As can be seen, for this case the application of the slope limiter has very little impact on the flow.

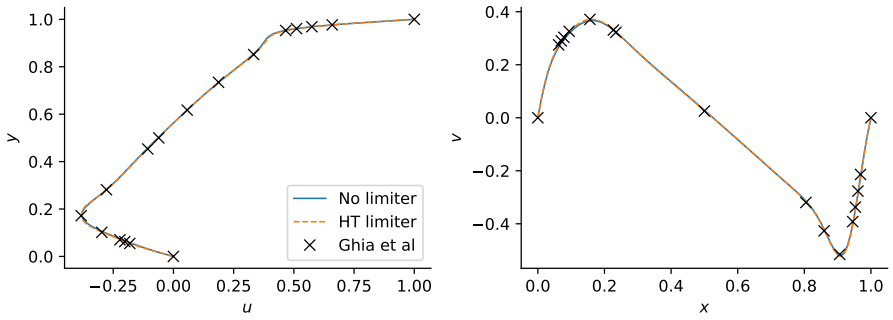


Figure A.2: Steady-state velocity components. Velocity in horizontal direction (u, on the left) and vertical direction (v, on the right). The horizontal component is extracted from a vertical cut through the centre of the domain, and the vertical component is extracted from a horizontal cut through the centre of the domain. The comparison values (X-marks) are from U. Ghia, K. N. Ghia, and Shin (1982).

A.3 Input file

An example input file for Ocellaris, used to compute the above results, is shown below. This is the input file for the slope limited case.

```
ocellaris:
    type: input
    version: 1.0

physical_properties:
    g: [0, 0]
    nu: 0.001
    rho: 1.0

mesh:
    type: Rectangle
    Nx: 40
    Ny: 40
    move:
    -   -1.944444444*pow(x[0], 3) + 2.916666667*pow(x[0], 2) +
        -0.9722222222*x[0] + -7.771561172e-16
    -   -0.7777777778*pow(x[1], 2) + 0.7777777778*x[1] + 0
```

```
time:
    dt: 0.01
    tmax: 30.0

boundary_conditions:
-   name: walls
    selector: code
    inside_code: on_boundary
    u:
        type: ConstantValue
        value: [0, 0]

-   name: lid  # overrides the "walls" above since it is defined later
    selector: code
    inside_code: on_boundary and x[1] >= 1.0 - 1e-8
    u:
        type: ConstantValue
        value: [1, 0]

solver:
    type: IPCS-A
    num_inner_iter: 2
    allowable_error_inner: 1e-3
    steady_velocity_stopping_criterion: 3.0e-5

slope_limiter:
    u:
        method: Componentwise
        comp_method: HierarchicalTaylor

output:
    prefix: lid_driven_cavity_flow
    xdmf_write_interval: 100

probes:
-   name: u-vel center
    type: LineProbe
    field: u0
    startpos: [0.5, 0]
    endpos: [0.5, 1]
    Npoints: 100
    file_name: uprobe.txt
    write_interval: 10

-   name: v-vel center
    type: LineProbe
    field: u1
    startpos: [0, 0.5]
    endpos: [1, 0.5]
    Npoints: 100
    file_name: vprobe.txt
    write_interval: 10
```

References

Ghia, U., Ghia, K. N., and Shin, C. T. (1982). "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method". *Journal of Computational Physics* 48.3, pp. 387–411.

Appendix B

Example: Taylor–Green vortex

B.1 Introduction

The 2D Taylor–Green vortex problem was shown in Paper I and Paper II. A domain with $(x, y) \in [0, 2]^2$, kinematic viscosity $\nu = 0.005$ and analytical solution

$$\begin{aligned} u &= -\sin(\pi y)\cos(\pi x)\exp(-2\pi^2\nu t) \\ v &= \sin(\pi x)\cos(\pi y)\exp(-2\pi^2\nu t) \\ p &= -{}^1\!/_4\,\rho(\cos 2\pi x + \cos 2\pi y)\exp(-4\pi^2\nu t) \end{aligned} \tag{B.1}$$

is simulated for $t \in [0, 1]$ with $\Delta t = 0.01$. Here the same setup is repeated with a larger selection of discretisations and a focus on assessing the numerical viscosity of the method. Unlike in the first paper, here the IPCS-A solver described in Paper II is used for efficiency reasons. Two inner iterations of the solver are run per time step.

B.2 Assessing the numerical viscosity

To assess the influence of the numerical dissipation, we start from the momentum equation for incompressible single-phase flow,

$$\rho\frac{\partial \boldsymbol{u}}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}\otimes\boldsymbol{u}) = \mu\nabla^2\boldsymbol{u} - \nabla p, \tag{B.2}$$

and multiply this by $\boldsymbol{u}$ to form the equation for the kinetic energy, $e_k = \frac{1}{2}\rho\,\boldsymbol{u}\cdot\boldsymbol{u}$,

$$\frac{\partial e_k}{\partial t} + \nabla\cdot(\boldsymbol{u}e_k) - \mu\boldsymbol{u}\cdot\nabla^2\boldsymbol{u} + \boldsymbol{u}\nabla p = 0. \tag{B.3}$$

The numerical dissipation, ϵ_n, can now be estimated and compared to the viscous dissipation, ϵ_v. The implementation here follows Castiglioni and Domaradzki (2015) and Schranner et al. (2015) and computes the numerical dissipation directly from the discrete velocity field. The numerical dissipation is the (negative) residual of the equation for kinetic energy (B.3), integrated over a sub-domain $\mathcal{T}_m$,

$$\frac{\partial E_k}{\partial t} + F_k - F_v + F_a + \epsilon_v = -\epsilon_n \tag{B.4}$$

where the volume-integrated kinetic energy is computed as

$$E_k = \int_{\mathcal{T}_m} \frac{1}{2}\rho\,\boldsymbol{u}\cdot\boldsymbol{u}\,\mathrm{d}\boldsymbol{x}, \tag{B.5}$$

and $\frac{\partial E_k}{\partial t}$ is then approximated by second order central differences,

$$\frac{\partial E_k}{\partial t} \approx \frac{E_k^{n+1} - E_k^{n-1}}{2\Delta t}. \tag{B.6}$$

The kinetic energy flux is

$$F_k = \int_{\mathcal{T}_m} \nabla \cdot (\boldsymbol{u} e_k) \, \mathrm{d}\boldsymbol{x}, \tag{B.7}$$

and the acoustic flux is

$$F_a = \int_{\mathcal{T}_m} \nabla \cdot (\boldsymbol{u} p) \, \mathrm{d}\boldsymbol{x}, \tag{B.8}$$

since the pressure term in equation (B.3) can be rewritten as

$$\boldsymbol{u} \cdot \nabla p = \nabla \cdot (\boldsymbol{u} p) - p \nabla \cdot \boldsymbol{u} = \nabla \cdot (\boldsymbol{u} p). \tag{B.9}$$

The viscous term from equation (B.3) is rewritten and split into a viscous flux, F_v, and a viscous dissipation, ϵ_v, from

$$\begin{aligned} \mu \boldsymbol{u} \cdot \nabla^2 \boldsymbol{u} &= \mu \nabla \cdot (\boldsymbol{u} \cdot \nabla \boldsymbol{u}) - \mu \nabla \boldsymbol{u} : \nabla \boldsymbol{u} \\ &= \nu \nabla^2 (\frac{1}{2} \rho \boldsymbol{u} \cdot \boldsymbol{u}) - \mu \nabla \boldsymbol{u} : \nabla \boldsymbol{u} \end{aligned} \tag{B.10}$$

leading to

$$F_v = \int_{\mathcal{T}_m} \frac{1}{2} \mu \nabla^2 (\boldsymbol{u} \cdot \boldsymbol{u}) \, \mathrm{d}\boldsymbol{x}, \tag{B.11}$$

$$\epsilon_v = \int_{\mathcal{T}_m} \mu \nabla \boldsymbol{u} : \nabla \boldsymbol{u} \, \mathrm{d}\boldsymbol{x}. \tag{B.12}$$

To assess the influence of the numerical dissipation on the simulation, the ratio of numerical to viscous dissipation is calculated, $r = \frac{\epsilon_n}{\epsilon_v}$. This ratio can be multiplied by the physical kinematic viscosity to obtain the equivalent numerical kinematic viscosity, $\nu_n = \nu r$.

The numerical dissipation is computed in a subset of the mesh cells, $\mathcal{T}_m \subseteq \mathcal{T}$. Multiple such sub-domains can be selected to study the appropriateness of the mesh resolution in different areas of interest, and this can then be used in a mesh refinement procedure, see Sun et al. (2017). Note that the selected sub-domain $\mathcal{T}_m$ must be sufficiently large for the residual to be mostly dissipative, otherwise dispersion errors may influence the results. The sub-domain should also contain areas of viscous shear for the comparison between numerical and physical dissipation to make sense. Also note that the effective Reynolds number, including both physical and numerical diffusion, can be written

$$\mathrm{Re}_{\mathrm{eff}} = \frac{\mathrm{Re}}{1 + r}. \tag{B.13}$$

For more details on estimating the numerical dissipation from the study of the discrete velocity field, see e.g. Cadieux, Sun, and Domaradzki (2017), Castiglioni and Domaradzki (2015), Fehn, Wall, and Kronbichler (2018), Komen et al. (2017), and Schranner et al. (2015).

B.3 Results

The sources of kinetic energy diffusion are computed in $\mathcal{T}_m = \mathcal{T}$ for each simulation, and the time-averaged value of the ratio of numerical to viscous dissipation is reported, $\bar{r}$. The numerical domain is divided into $N \times N$ squares which are each subdivided into two triangles. The results for a set of numerical discretisations are shown in table B.1. In the table P2P1 means that DG elements with bi-quadratic polynomials are used for the velocity while bi-linear DG elements are used for the pressure. As can be seen the numerical diffusion decreases with decreasing element sizes, and the combined numerical and physical viscosities never become negative.

It is clear from table B.1 that the equal-order discretisations performs significantly worse in terms of numerical viscosity compared to the P2P1 and P3P2 discretisations. The choice of equal discretisation orders for the velocity and pressure fields violates the inf–sup/LBB criterion, even when the solver employed is a pressure-correction solver, and this leads to oscillations in the pressure field (Codina 2001; Guermond and Quartapelle 1998). When comparing the terms in equation (B.4) it is clear that it is the pressure-related acoustic flux term, F_a, that is the problem, see figure B.1. It can also be seen in figure B.2 that the velocity converges with the expected order, while the pressure does not for the equal orders.

Table B.1: Time averaged numerical diffusion ratio, ϵ_n in percent of ϵ_v.

DG elements	N = 10	N = 20	N = 30	N = 40
P2P1	52.3%	16.0%	7.4%	4.3%
P2P2	94.9%	29.6%	13.6%	7.4%
P3P2	-1.4%	-0.3%	-0.1%	-0.1%
P3P3	-26.6%	-6.4%	-2.5%	-1.2%

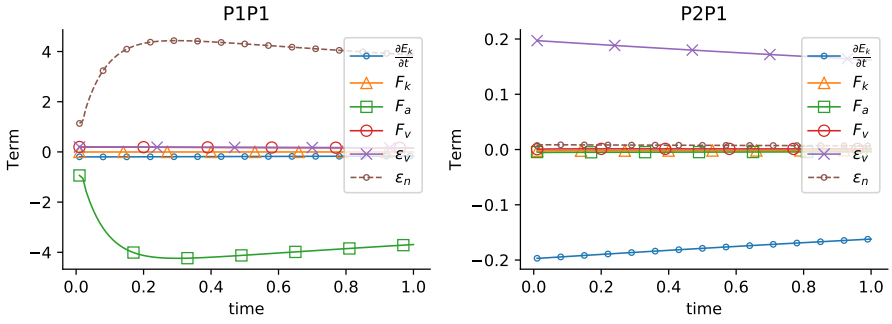


Figure B.1: Terms from equation (B.4) for P1P1 and P2P1 discretisations at $N = 40$.

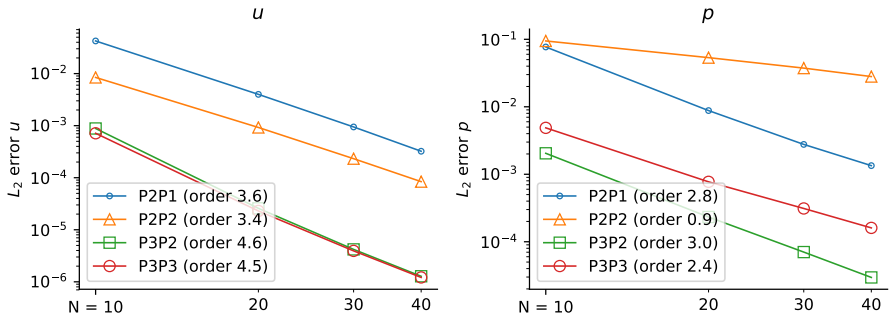


Figure B.2: L_2 error of the velocity and pressure fields. The convergence orders shown in parentheses are the average for the line. The observed super-convergence for some results is due to the highly regular mesh.

To investigate the accuracy of the computed numerical dissipation from equation (B.4), three additional (physical) kinematic viscosities were investigated for the P2P1 discretisation. The results can be found in table B.2. As can be seen the estimates seem relatively accurate when comparing the results from $\nu = 0.01$ and $\nu = 0.001$ (i.e. there is approximately a factor 10 difference in ϵ_n/ϵ_v for the finer meshes), but the results for $\nu = 0.02$ and $\nu = 0.005$ are much closer to the results for $\nu = 0.01$ than would be appropriate for their size. Still it seems that the computed numerical viscosity works well as an indicator to assess the diffusive influence of the numerical scheme on the results.

Table B.2: Time averaged numerical diffusion ratio, ϵ_n in percent of ϵ_v, for four different kinematic viscosities, using P2P1 elements.

Viscosity	N = 10	N = 20	N = 30	N = 40
$\nu = 0.02$	40.7%	11.4%	5.1%	2.7%
$\nu = 0.01$	44.7%	12.5%	5.7%	3.2%
$\nu = 0.005$	52.3%	16.0%	7.4%	4.3%
$\nu = 0.001$	109.6%	74.9%	44.4%	28.5%

Finally, to see the impact of the choice of solver and the slope limiter, two more test are presented in table B.3, still compared to the P2P1 discretisation. The IPCS-A solver used for the other results is compared to the Coupled solver which does not have any splitting errors due to using a direct sparse solver to solve for the velocity and pressure at the same time. Also included is the IPCS-A solver with the hierarchical Taylor-polynomial based velocity-field slope limiter. As can be seen the differences are marginal except for the coarsest resolution where the limiter detects unwanted wiggles in the velocity field and proceeds to smoothen out the slopes, introducing some diffusion. For the better resolved test cases the limiter does not influence the field in a noticeable way.

Table B.3: Time averaged numerical diffusion ratio, ϵ_n in percent of ϵ_v, for three different solver options.

Solver	N = 10	N = 20	N = 30	N = 40
IPSC-A	52.3%	16.0%	7.4%	4.3%
Coupled	52.4%	16.0%	7.4%	4.2%
IPSC-A + limiter	84.8%	17.4%	7.6%	4.3%

B.4 Input file

An example input file for Ocellaris, used to compute the above results, is shown below. This is the P2P1 input file for $\nu = 0.05$ and $N = 40$.

```
ocellaris:
    type: input
    version: 1.0

physical_properties:
    g: [0, 0]
    nu: 0.005
    rho: 1.0

mesh:
    type: Rectangle
    Nx: 40
    Ny: 40
    endx: 2
    endy: 2

boundary_conditions:
-   name: walls
    selector: code
    inside_code: on_boundary
    u:
        type: CppCodedValue
        cpp_code:
        -   -sin(pi*x[1]) * cos(pi*x[0]) * exp(-2*pi*pi*nu*t)
        -    sin(pi*x[0]) * cos(pi*x[1]) * exp(-2*pi*pi*nu*t)

initial_conditions:
    up0:
        cpp_code: -sin(pi*x[1])*cos(pi*x[0])*exp(-2*pi*pi*nu*t)
    up1:
        cpp_code:  sin(pi*x[0])*cos(pi*x[1])*exp(-2*pi*pi*nu*t)
    upp0:
        cpp_code: -sin(pi*x[1])*cos(pi*x[0])*exp(-2*pi*pi*nu*(t-dt))
    upp1:
        cpp_code:  sin(pi*x[0])*cos(pi*x[1])*exp(-2*pi*pi*nu*(t-dt))
    p:
        cpp_code: -(cos(2*pi*x[0]) + cos(2*pi*x[1])) * exp(-4*pi*pi*nu*t)/4

time:
    dt: 0.01
```

```
    tmax: 1.0

output:
    prefix: taylor_green_out

solver:
    type: IPSC-A
    num_inner_iter: 2
    polynomial_degree_velocity: 2
    polynomial_degree_pressure: 1
    function_space_velocity: DG
    function_space_pressure: DG

slope_limiter:
    u:  # Only active when this method = Componentwise
        method: None
        comp_method: HierarchicalTaylor
        skip_boundaries: ["walls"]

hooks:
    post_timestep:

    -   name: Compute l2 error
        code: |
            cppu = "-sin(pi*x[1])*cos(pi*x[0])*exp(-2*pi*pi*nu*t)"
            cppv = "sin(pi*x[0])*cos(pi*x[1])*exp(-2*pi*pi*nu*t)"
            cppp = "-(cos(2*pi*x[0]) + cos(2*pi*x[1])) * exp(-4*pi*pi*nu*t)/4"
            ua = Expression(cppu, nu=nu, t=t, degree=7)
            va = Expression(cppv, nu=nu, t=t, degree=7)
            pa = Expression(cppp, nu=nu, t=t, degree=7)

            def calc_err(f_num, e_ana):
                V = f_num.function_space()
                f_ana = project(e_ana, V)
                f_err = dolfin.Function(V)
                f_err.vector()[:] = f_ana.vector()[:] - f_num.vector()[:]
                return dolfin.norm(f_err) / dolfin.norm(f_ana)

            simulation.reporting.report_timestep_value('errU', calc_err(u0, ua))
            simulation.reporting.report_timestep_value('errV', calc_err(u1, va))
            simulation.reporting.report_timestep_value('errP', calc_err(p, pa))

    -   name: Assess the numerical viscosity
        code: |
            ek = 0.5 * rho * dot(u, u)
            Ek = assemble(ek * dx)
            Fk = assemble(div(u * ek) * dx)
            Fa = assemble(div(u * p) * dx)
            Fv = assemble(nu * div(grad(ek)) * dx)
            eps_v = assemble(mu * inner(nabla_grad(u), nabla_grad(u)) * dx)
            simulation.reporting.report_timestep_value('MyEk', Ek)
            simulation.reporting.report_timestep_value('MyFk', Fk)
            simulation.reporting.report_timestep_value('MyFa', Fa)
            simulation.reporting.report_timestep_value('MyFv', Fv)
            simulation.reporting.report_timestep_value('MyEpsV', eps_v)
```

References

Cadieux, F., Sun, G., and Domaradzki, J. A. (2017). "Effects of numerical dissipation on the interpretation of simulation results in computational fluid dynamics". *Computers & Fluids* 154, pp. 256–272.

Castiglioni, G. and Domaradzki, J. A. (2015). "A numerical dissipation rate and viscosity in flow simulations with realistic geometry using low-order compressible Navier–Stokes solvers". *Computers & Fluids* 119, pp. 37–46.

Codina, R. (2001). "Pressure Stability in Fractional Step Finite Element Methods for Incompressible Flows". *Journal of Computational Physics* 170.1, pp. 112–140.

Fehn, N., Wall, W. A., and Kronbichler, M. (2018). "Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows". *Journal of Computational Physics* 372, pp. 667–693.

Guermond, J.-L. and Quartapelle, L. (1998). "On stability and convergence of projection methods based on pressure Poisson equation". *International Journal for Numerical Methods in Fluids* 26.9, pp. 1039–1053.

Komen, E. M. J. et al. (2017). "A quantification method for numerical dissipation in quasi-DNS and under-resolved DNS, and effects of numerical dissipation in quasi-DNS and under-resolved DNS of turbulent channel flows". *Journal of Computational Physics* 345, pp. 565–595.

Schranner, F. S. et al. (2015). "Assessing the numerical dissipation rate and viscosity in numerical simulations of fluid flows". *Computers & Fluids* 114, pp. 84–97.

Sun, G. et al. (2017). "Assessing accuracy of CFD simulations through quantification of a numerical dissipation rate". In: *Proceedings of TSFP-10 (2017) Chicago*. Chicago, USA, p. 6.

Appendix C

Example: flow around 2D cylinder

C.1 Introduction

In this example the Turek–Schäfer CFD benchmark case DFG 2D-2 Schäfer et al. (1996) is studied with a focus on the interaction between the viscous shear at the cylinder and the slope limiter. The geometry is shown in figure C.1.

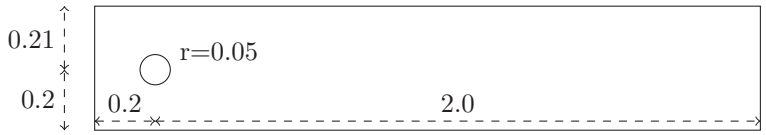


Figure C.1: The DFG 2D-2 benchmark geometry.

The velocity at the inlet (the leftmost vertical wall) is parabolic with zero velocity at the top and bottom and a maximum horizontal velocity of $u_{\max} = 1.5$ in the center. When taking the origin to be the lower left corner and the x and y axes pointing right and up respectively, the inlet velocity is

$$u = 2\, u_{\max}\; y\,(0.41 - y)\, 0.41^{-2}, \tag{C.1}$$

$$v = 0. \tag{C.2}$$

Just like in Paper III, a forcing zone is used towards the outlet, starting at the horizontal centre of the domain and ramping up to a maximum penalty of 10 at the outlet, forcing the computed velocity towards the inlet velocity. In this way the inlet velocity can be used as a Dirichlet boundary condition for the whole domain, except for the circular cylinder where $u = v = 0$.

The average inlet velocity is $u_{\text{avg}} = 1.0$, giving a Reynolds number of

$$\text{Re} = \frac{u_{\text{avg}}\, 2r}{\nu} = \frac{1.0 \cdot 0.1}{0.001} = 100, \tag{C.3}$$

a regime where periodic vortex shedding will be triggered by the cylinder, causing a vortex street to form behind the cylinder.

It should be noted that the following results are not the product of a thorough investigation. Only one mesh has been tested and no particular tuning of any of the parameters has been done. First order geometry is used for the mesh (see figure C.4), so the curvature of the cylinder is described with linear elements. Ocellaris, through FEniCS, supports higher order geometry, but the parts of the code that directly work with cell geometries and degrees of freedom (only the slope limiter in this instance) would need to be thoroughly investigated and validated for use with higher-order geometry, and this has not been done.

C.2 Results

In addition to showing the application of Ocellaris to a common benchmark problem, this results section will focus on the problems caused by applying the slope limiter close to the boundaries. As described in section 1.7, since no neighbour information is available on the outside of the domain, the limiter will tend to over-flatten slopes of elements close to the boundaries. The results are hence reported for a simulation with the slope limiter disabled, a simulation with the slope limiter turned on, and lastly a simulation with the slope limiter enabled in cells not touching a boundary.

In figure C.2 the results are compared to the benchmark values $C_D = 3.2200$, $C_L = 0.9859$ and $St = 0.30188$. In general the non-limited simulation performs well, the partially limited simulation is less good and the fully limited solution is worst. The fact that the partially limited simulation is not as good as non-limited solution suggests further studies are needed and that perhaps the development of a troubled cell indicator that takes into account the position of the free surface could be integrated in order to restrict the limiter only to areas where it is needed for stability of the solver.

The the peak values of the total force on the cylinder in x and y direction are used to compute the coefficients and the Strouhal number. The frequency f is the inverse of the time between peaks in the lift force. The formulas used are

$$C_D = {}^{(2F_x)}/_{u_{\text{avg}}^2}, \tag{C.4}$$

$$C_L = {}^{(2F_y)}/_{u_{\text{avg}}^2}, \tag{C.5}$$

$$St = {}^{(2rf)}/_{u_{\text{avg}}}. \tag{C.6}$$

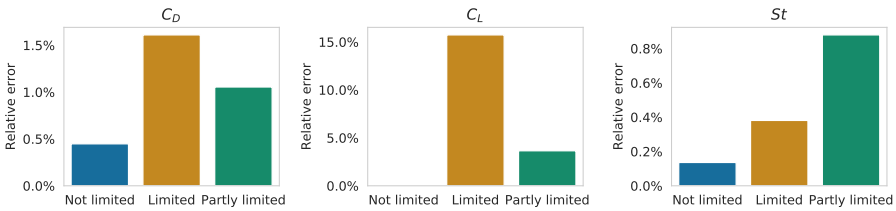


Figure C.2: The relative error, $|{}^{(C - C_{\text{ref}})}/_{C_{\text{ref}}}|$, of the drag and lift coefficients and the Strouhal number are shown separately. The values are computed from the peaks in the time series of combined viscous and pressure forces.

Lastly, the numerical dissipation is computed for the area close to the cylinder (within three radiuses of the cylinder center in both x and y directions). The results show that the velocity slope limiter induces numerical viscosity. This is reasonable and as expected. The numerical viscosity is presented as ϵ_n in percent of ϵ_v, as explained in the previous appendix. The results are shown in figure C.3. The numerical viscosity stabilises quite rapidly (the same is true for the drag and lift coefficients, not shown here).

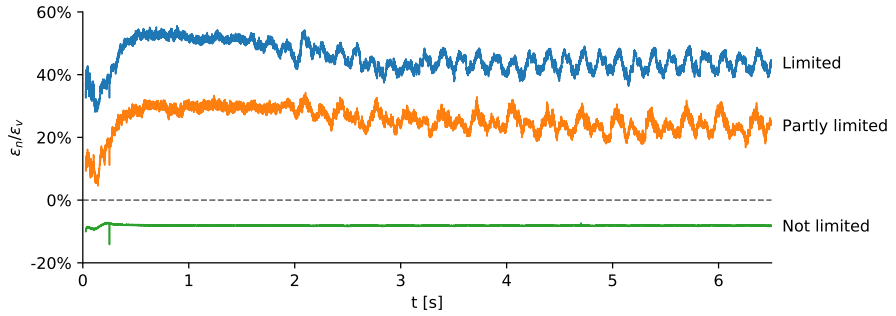


Figure C.3: Numerical viscosity near the cylinder.

C.3 Input file

The Gmsh (Geuzaine and Remacle 2009) geometry file is attached below

```
//////////////////////////////////////////
// Created on 2018-04-20 by Tormod Landet
// Surface piering bottom mounted cylinder
// in a wave flume
SetFactory("OpenCASCADE");


//////////////////////////////////////////
// Parameters

// Geometrical parameters
DefineConstant[ R  = {  0.05, Name "Parameters/Cylinder radius" } ];
DefineConstant[ L  = {  2.20, Name "Parameters/Domain length" } ];
DefineConstant[ B  = {  0.41, Name "Parameters/Domain breadth" } ];
DefineConstant[ Cx = {  0.20, Name "Parameters/Cylinder dist from inlet" } ];
DefineConstant[ Cy = {  0.20, Name "Parameters/Cylinder dist from bottom" } ];

// Mesh parameters
Q = 6; // Coursener, lower value gives finer mesh
DefineConstant[ lc_fine   = {0.001 * Q, Name "Parameters/LC fine" } ];
DefineConstant[ lc_med    = {0.002 * Q, Name "Parameters/LC medium" } ];
DefineConstant[ lc_course = {0.010 * Q, Name "Parameters/LC course" } ];


//////////////////////////////////////////
// Geometry

// The domain
domain = newv; Rectangle(domain) = {0, 0, 0,   L, B};

// The cylinder
cylinder = newv; Disk(cylinder) = {Cx, Cy, 0, R};

// Delete cylinder from domain
domain_new = newv;
```

```
BooleanDifference(domain_new) =
  { Surface{domain}; Delete; }{ Surface{cylinder}; Delete; };
domain = domain_new;

////////////////////////////////////////////
// Mesh cell size fields:

// Vortex street
Field[80] = MathEval;
Field[80].F =
  Sprintf("(((x - %g)/%g)^2 + ((y - %g)/%g)^2)^0.5", Cx + 6*R, 8*R, Cy, 4*R);
Field[88] = Threshold;
Field[88].DistMax = 3;
Field[88].DistMin =  1;
Field[88].IField = 80;
Field[88].LcMax = lc_course;
Field[88].LcMin = lc_med;

// Overall mesh grading
Field[90] = MathEval;
Field[90].F =
  Sprintf("(((x - %g)/%g)^2 + ((y - %g)/%g)^2)^0.5", Cx + R, 1.7*R, Cy, 1.2*R);
Field[99] = Threshold;
Field[99].DistMax = 15;
Field[99].DistMin =  1;
Field[99].IField = 90;
Field[99].LcMax = lc_course;
Field[99].LcMin = lc_fine;

// The resulting mesh size field is the minimum of the above fields
Field[100] = Min;
Field[100].FieldsList = {88, 99};
Background Field = 100;


////////////////////////////////////////////
// Physical domains

Physical Volume(100) = { domain }; // The fluid domain


// The end, code below is added by gmsh GUI
////////////////////////////////////////////
```

The mesh generated from the above input file can be seen in figure C.4. There are approximately 9000 cells.

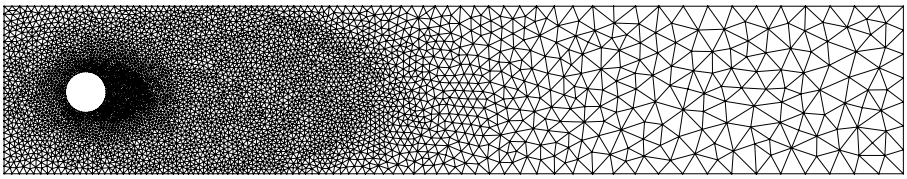


Figure C.4: The generated mesh.

An example input file for Ocellaris, used to compute the above results, is shown below. This is the input file for the partially slope limited case.

```
ocellaris:
    type: input
    version: 1.0

metadata:
    author: Tormod Landet
    date: 2019-11-02
    description: |
        Flow around a 2D cylinder, CFD Benchmarking project
        DFG benchmark 2D-2, Re100, periodic
        http://www.featflow.de/en/benchmarks/cfdbenchmarking/
        flow/dfg_benchmark2_re100.html

user_code:
    constants:
        B: 0.41
        R: 0.05
        L: 2.20
        Cx: 0.2
        Cy: 0.2
        MIN_DT: 0.00003
        MAX_DT: 0.01
        CRIT_CO: 0.03

physical_properties:
    g: [0, 0]
    nu: 0.001
    rho: 1.0

mesh:
    type: meshio
    mesh_file: ../cylinder2D_only_triangles_Q6.msh


fields:
-   name: inletvelocity
    type: VectorField
    variable_name: u
    cpp_code:
    -   "4 * 1.5 * x[1] * (B - x[1]) / B / B"
    -   "0.0"

-   name: outlet zone
    type: ScalarField
    variable_name: beta
    cpp_code: |
        [&]() {
            double dz0 = L / 2;
            double dz1 = L;
            if (x[0] < dz0) {
                return 0.0;
            } else if (x[0] > dz1) {
                return 1.0;
            } else {
```

```
                return pow((x[0] - dz0)/(dz1 - dz0), 2);
            }
        }()

boundary_conditions:
-   name: InletOutlet
    selector: code
    inside_code: on_boundary
    u0:
        type: FieldFunction
        function: inletvelocity/u0

    u1:
        type: ConstantValue
        value: 0

-   name: TopBottom
    selector: code
    inside_code: on_boundary and (x[1] < 1e-6 or x[1] > B - 1e-6)
    u0:
        type: ConstantValue
        value: 0
    u1:
        type: ConstantValue
        value: 0

-   name: Cylinder
    selector: code
    inside_code: >-
        on_boundary and x[0] > Cx - 1.1*R and x[0] < Cx + 1.1*R
        and x[1] > Cy - 1.1*R and x[1] < Cy + 1.1*R
    u0:
        type: ConstantValue
        value: 0
    u1:
        type: ConstantValue
        value: 0

forcing_zones:
-   name: outlet velocity damping
    type: MomentumForcing
    zone: outlet zone/beta
    target: inletvelocity/u
    penalty: 10
    plot: yes

time:
    dt: 0.001
    tmax: 30.0

solver:
    type: IPCS-A
    num_inner_iter: py$ 10 if it < 10 else 2

slope_limiter:
    u:  # Only active when method is "Componentwise"
        method: Componentwise
```

```
        comp_method: HierarchicalTaylor
        skip_boundaries: ["all"]

output:
    prefix: flow_past_cylinder
    xdmf_write_interval: 20

probes:
    -   name: pressure_probes
        enabled: yes
        type: PointProbe
        probe_points:
        -   py$ ['p', 'pressure_leading', Cx - R * 1.0001, Cy]
        -   py$ ['p', 'pressure_trailing', Cx + R * 1.0001, Cy]

hooks:
    post_timestep:
    -   name: Forces on the cylinder
        enabled: yes
        code: |
            u = up_conv # the velocity before slope limiting
            if not 'forceforms' in hook_data:
                cyl_id = boundary_by_name['Cylinder'].mark_id
                cyl_ds = ds(cyl_id)
                n = FacetNormal(mesh)
                ff = hook_data['forceforms'] = {}
                Tp = - p * n
                Tv = 2 * mu * dot(sym(nabla_grad(u)), n)
                ff['Fpx'] = Form(dot(Tp, as_vector([1, 0])) * cyl_ds)
                ff['Fpy'] = Form(dot(Tp, as_vector([0, 1])) * cyl_ds)
                ff['Fvx'] = Form(dot(Tv, as_vector([1, 0])) * cyl_ds)
                ff['Fvy'] = Form(dot(Tv, as_vector([0, 1])) * cyl_ds)
            for name, form in hook_data['forceforms'].items():
                val = assemble(form)
                simulation.reporting.report_timestep_value(name, val)

    -   name: Adjust timestep
        enabled: yes
        code: |
            # Cell and facet based Courant numbers
            Co = simulation.reporting.timestep_xy_reports.get('Co', [0])
            Cof = simulation.reporting.timestep_xy_reports.get('Cof_max', [0])

            # Max values in a window (to avoid changing time step too often)
            if len(Co) > 15:
                window_Co = numpy.max(Co[-10:])
                window_Cof = numpy.max(Cof[-10:])
            else:
                window_Co = 1e10
                window_Cof = 1e10

            dt_lower = simulation.dt / 2
            dt_higher = simulation.dt * 2

            new_dt = None
            if (max(Co[-1], Cof[-1]) > CRIT_CO
                and dt_lower - MIN_DT > 1e-4):
```

```
            new_dt = dt_lower
        elif (max(window_Co, window_Cof) < CRIT_CO / 6
              and dt_higher - MAX_DT < 1e-4):
            new_dt = dt_higher

        if new_dt is not None:
            simulation.log.info(
                'Changing dt from %.5f to %.5f' % (dt, new_dt)
            )
            simulation.input.set_value('time/dt', new_dt)

-   name: Assess the numerical viscosity
    code: |
        domains = [
                    ("all", "1.0"),
                    ("nearcyl",
                     "abs(x[0] - Cx) < 3 * R and abs(x[1] - Cy) < 3 * R"),
                    ("street",
                     "abs(x[0] - Cx - 7*R) < R * 5 "
                     "and abs(x[1] - Cy) < 3 * R")
        ]
        u = up_conv # the velocity before slope limiting
        for domain, domain_cpp in domains:
            omega = Expression(domain_cpp, R=R, Cx=Cx, Cy=Cy, degree=1)
            ek = 0.5 * rho * dot(u, u)

            # Compute terms of the kinetic energy equation
            Ek = assemble(ek * omega * dx)
            Fk = assemble(div(u * ek) * omega * dx)
            Fa = assemble(div(u * p) * omega * dx)
            Fv = assemble(nu * div(grad(ek)) * omega * dx)
            eps_v = assemble(
                mu * inner(nabla_grad(u), nabla_grad(u)) * omega * dx
            )

            # Add computed numbers to logs
            report = simulation.reporting.report_timestep_value
            report('Ek_%s' % domain, Ek)
            report('Fk_%s' % domain, Fk)
            report('Fa_%s' % domain, Fa)
            report('Fv_%s' % domain, Fv)
            report('EpsV_%s' % domain, eps_v)
```

References

Geuzaine, C. and Remacle, J.-F. (Sept. 10, 2009). "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". *International Journal for Numerical Methods in Engineering* 79.11, pp. 1309–1331.

Schäfer, M. et al. (1996). "Benchmark Computations of Laminar Flow Around a Cylinder". In: *Flow Simulation with High-Performance Computers II: DFG Priority Research Programme Results 1993–1995*. Ed. by Hirschel, E. H. Notes on Numerical Fluid Mechanics (NNFM). Wiesbaden: Vieweg+Teubner Verlag, pp. 547–566.