A Level Set Discontinuous Galerkin Method for Free Surface Flows

— and Water-Wave Modeling

Jesper Grooss

Ph.D. Thesis, February 2005
Informatics and Mathematical Modelling
Technical University of Denmark
Kongens Lyngby, Denmark

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kgs. Lyngby, Denmark
Phone +45 4525 3351, Fax +45 4588 2673
reception@imm.dtu.dk
www.imm.dtu.dk

Thesis submitted February 24, 2005.
Typeset using LaTeX.

IMM-PHD-2005-145
ISSN 0909-3192
ISBN 87-643-0016-1

Preface

This thesis is submitted to the Technical University of Denmark in partial fulfillment of the requirements for the degree of Doctor of Philosophy. The work was performed at Informatics and Mathematical Modelling, Technical University of Denmark, during the period September 2001 to February 2005, and was sponsored by the university.

The work has been supervised by Professor Per Grove Thomsen, Informatics and Mathematical Modelling, and as secondary supervisor Professor Jens Nørkær Sørensen, Department of Mechanical Engineering, both Technical University of Denmark. I would like to thank my supervisors for their support, friendliness and relaxed attitude. The subject of my work was fairly new to all of us, and I was given the freedom to explore any subject that I found interesting, and shape my work as I found appropriate.

Special thanks goes to Professor Jan S. Hesthaven, Brown University, USA, whom in many aspects has been my scientific supervisor. He kindly hosted my six month stay at Brown University during fall-winter 2002-2003, where I got to know Professor Hesthaven, his colleagues, students and family. I learned a lot from him, and have always been met with sincere interest and professionalism.

I also thank Michael Jacobsen, Jan Marthedal Rasmussen and Toke Jensen. With the two latter I shared, in turn, a DTU office, and Michael always showed a genuine interest in my work and especially my results. They are good friends of mine, and have always had time to listen to my problems, my victories and defeats, academic as well as personal.

I thank both Allan P. Engsig-Karup and Jan Marthedal Rasmussen for good advice and for proofreading this thesis.

And last but not least, many thanks to my family, friends and colleagues for patience, encouragement and support.

Kongens Lyngby, February 24, 2005

Jesper Grooss

Abstract

We present a discontinuous Galerkin method on a fully unstructured grid for the modeling of unsteady incompressible fluid flows with free surfaces. The surface is modeled by a level set technique.

We describe the discontinuous Galerkin method in general, and its application to the flow equations. The discontinuous Galerkin method is based on triangular elements, giving maximum flexibility in the handling of complex domains. A high order nodal basis is used to enable local high order and is well suited for problems with many scales and wave lengths.

The use of high-order methods for advancing the unsteady equations in time are discussed. We investigate theory of differential algebraic equations, and connect the theory to current methods for solving the unsteady fluid flow equations. We explore the use of a semi-implicit spectral deferred correction method having potential to achieve high temporal order. The deferred correction method is applied on the fluid flow equations and show good results in periodic domains.

We describe the design of a level set method for the free surface modeling. The level set utilize the high order accurate discontinuous Galerkin method fully and represent smooth surfaces very accurately. We present techniques for reinitialization, and outline the strengths and weaknesses of the level set method.

Through a few numerical tests, the robustness and versatility of the proposed scheme is confirmed.

Resumé

Til modellering af de ikke-stationære ikke-komprimerbare strømnings-ligninger med frie overflader, præsenterer vi en diskontinuert Galerkin metode på et fuldt ustruktureret net. Overfladen modelleres med en level set teknik.

Vi beskriver den diskontinuerte Galerkin metode generelt, og dens brug på strømnings-ligningerne. Den diskontinuerte Galerkin metode er baseret på trekantede elementer, og giver maximal fleksibilitet til at håndtere komplekse domæner. En høj ordens nodal basis giver lokal høj ordens nøjagtighed, og er velegnet til problemer med mange skalaer og bølge-længder.

Vi diskuterer brugen af højere ordens metoder til tids-integration af de ikke-stationære strømningsligninger. Teori fra differential-algebraiske ligninger inddrages, og forbindes til nuværende metoder til løsning af strømningsligningerne. Vi undersøger brugen af en semi-implicit spectral deferred correction metode, som potentielt kan tids-integrere strømningsligningerne med høj ordens nøjagtighed. Vi anvender metoden på strømningsligningerne og viser gode resultater for periodiske problemer.

Vi beskriver en level set metode til modellering af den frie overflade. Level set metoden udnytter den høje nøjagtighed fra den diskontinuerte Galerkin metode. Vi præsenterer teknikker til at reinitialisere level settet, og beskriver level set teknikkens styrker og svagheder.

Et antal numeriske test bekræfter robustheden og brugbarheden af den foreliggende metode.

Contents

CHAPTER 1

Introduction to Free Surface Flow

Παντα ρει και ουδεν μενει[1]
— HERACLITUS ~535–475 BC

In maritime engineering one is often interested in the modeling and prediction of phenomena involving free surfaces. Examples are:

- Wave impact on off-shore structures: When a wave hits e.g. an off-shore oil-rig, we would want to know what structural forces are involved, which part of the oil-rig structure gets the highest load, and how the rig should be designed to minimize this load and withstand the forces.

- Green water load on ships: When the wave height exceeds the height of a ship's side or bow and water runs over the deck, it is called green water. Modern carriers are designed to not turn over or sink, even for large amounts of water on deck, and the water will run off the deck when the deck again exceeds the water level. However, the amount of water and the resulting forces can be huge and can cause damage to equipment or cargo on the deck.

- Tank Sloshing: When a water or gasoline truck with a partly filled tank starts turning or braking, the water will move back and forth within the tank. Apart from being unpleasant for the driver, it makes the truck difficult to steer and may even overturn the truck. For a given tank and degree of filling, the water will move back and forth with a natural frequency, and we must make sure not to exert forces to the tank in this frequency and thereby possibly amplify the water motion.

[1] Everything flows, nothing stands still

For ships carrying oil or other liquids in huge tanks, the sloshing liquid in partly filled tanks can produce strong forces on the side of the tank, which may produce structural damages to the tank. A tank designed to withstand the pressure when full, may not withstand the pressure from the sloshing in a partly filled tank.

- Surface piercing pipeline/cylinder: Consider a pipeline arising from the bottom of the sea and connecting to an oil-rig above the sea level. This pipeline will be exerted to forces from the ocean current as well as the waves. At certain wave frequencies and certain current velocities, this may exert the pipeline in its natural frequency and start an oscillation which may eventually damage the pipeline.

Other examples include dam break problems, and surf zone dynamics.

Such modeling efforts are not only complicated due to the flow, but also by the need to accurately account for the free surface and for the fluid-structure interaction problems. Often the structure is complicated, and geometrically complex domains are therefore needed. Furthermore, these types of problems often involve a variety of length scales, from an incoming wave length of several meters to the flow of microscopic scale around the structures, putting rather severe requirements on suitable computational techniques.

1.1 Free Surface Modeling

The flow for these kind of problems are typically modeled by the *incompressible Navier Stokes* equations, which we shall present in Chapter 2. The water can in most practical cases be considered incompressible, making the incompressible Navier Stokes equations suitable.

Most flow simulations consider only one type of fluid, i.e., only water or only air. For free surface problems, we have both air and water. From a practical point of view, the water part is by far the most important, since the load from the air is small compared to that of the water. When modeling the water part only, then the computational domain changes in time, as the free surface moves. We may say that the computational domain becomes a part of the solution. Another option is to let the computational domain include both the air and water part, simulate both fluids, and in some way capture or keep track of the interface between the two fluids - the surface.

There are many strategies for incorporating a free surface in the flow simulation. The most popular include *moving (Lagrangian) grid techniques*, *marker and cell (MAC)*, *volume of fluid (VOF)*, and *level set methods*. We shall give a short overview of the different methods here, and outline their strengths and weaknesses.

Moving grid techniques model the water part only, and adapt the grid to the domain of the water as it changes. It is necessary to update the grid continuously, and it may also be necessary once in a while to re-grid the entire domain. The grid becomes time dependent, and interpolation is needed to transfer a solution on a grid corresponding to time t_i onto the grid corresponding to the next instance of time t_{i+1}. The technique gives the surface position accurately and surface derivatives as normal and curvature can also be computed accurately. The techniques do not easily handle droplets or bubbles inside the main fluid. Merging and breakup of the surface, e.g., when two water droplets collide and merge into one big droplet, is difficult to handle, since one would need to determine the time at which it happens, and update the grid in a domain which has changed topology, e.g., from two independent grids into one common grid. For an example, see [47].

The MAC method introduced in [30] includes the evolution of a large number of *marker particles*. A computational cell is defined to be filled with water when it contains at least one marker particle. Evolution of the surface is computed by moving the markers with the flow. Different extensions exists, that, e.g., only evolves particles within some distance of the surface, called *surface marker methods*. The MAC method automatically handles breakup and merging of volumes implicitly, however the evolution of the particles are computationally expensive, especially in 3D. Redistribution of the particles may be necessary once in a while, to avoid sudden void parts to build up in the middle of the water. In [51] are examples describing limitations and advantages in more detail.

The VOF method introduced in [38], perhaps the most widely used technique for such problems, includes a *volume fluid fraction* in each cell. The cell is filled or emptied depending on the flow and the volume fluid fraction in neighboring cells. The VOF methods can handle merging and breakup of the surface implicitly, and in a conservative setup has a high degree of mass conservation.

Both the MAC and VOF methods have a disadvantage related to surface derivatives as normal and curvature, which can be quite inaccurate, though more recent VOF methods have improved on that point.

Level set methods [54, 64, 66] define the surface as the zero contour of a scalar function. This handles merging and breakup of the surface implicitly. While it may suffer from mass loss, especially for very non-smooth surfaces and at low resolution, it has sub-element accuracy of the surface position, i.e., the surface position can be determined accurately within the element. It can compute the surface derivatives to high precision, and it maintains volume shapes.

1.2 The Present Work

In this thesis we shall discuss the development of high-order accurate discontinuous Galerkin (DG) methods for the modeling of unsteady incompressible fluid flows with free surfaces. The DG method utilizes a fully unstructured grid based on triangular elements, viable of handling complicated geometries. A high-order nodal basis is used to enable local high order and is well suited for problems with many scales and wave lengths.

The flow shall be described by the incompressible two-fluid Navier Stokes equations, which we present in Chapter 2. To represent the free surface, a level set approach is used, since the level set method can take advantage of the high-order accurate DG method and provide high sub-element accuracy of the surface position, and highly accurate surface derivatives. The level set is mostly smooth and is represented well by high order methods.

We shall likewise discuss the use of high-order schemes for advancing the unsteady equations in time. As we present in Chapter 4, for computational reasons, the most common approach when solving the incompressible Navier Stokes equations is to decouple the evaluation of the pressure and the velocities, and furthermore treat the linear terms implicitly and the non-linear terms explicitly. The errors introduced by the decoupling procedure, called time splitting errors, must be given special attention in order to retain accuracy. In an attempt to address this, we explore the use of a *semi-implicit spectral deferred correction method*, following the work in [19, 48], although recast to work on the physical variables and therefore suited to free surface flow problems. Examples will show the method to be very flexible and with potential to achieve high temporal order.

Though the emphasis in this work is on the two fluids water and air, we shall point out that the techniques are applicable to any pair of immiscible fluids. Furthermore, it may fairly easy be extended to three fluid flow, by introducing yet another level set variable.

The goal of this work has been to explore, test and develop methods and strategies for solving free surface flow problems. The long term objective is to produce a new generation of a free surface flow solver, replacing an existing one present at the Technical University of Denmark. The focus is therefore on practical aspects, thus we will present existing and new methods, but not provide many proofs of their properties, basing our validation mostly on numerical tests.

The main contribution to existing work in this thesis are: Application of the DG method to the two-fluid incompressible Navier Stokes equations. Development of high order temporal scheme to integrate the incompressible Navier Stokes equations in time. Adaption and application of the DG method to the level set approach and techniques for handling the level set, including band-limiting, boundary conditions and reinitialization.

Some of the results of this work is also presented in [25].

1.2.1 Outline of Thesis

Apart from the present introduction, the thesis consists of three main parts devoted each to a main subject and thereafter a small numerical/validation part. However, before the three main parts, Chapter 2 will set the stage by describing the equations for the two-dimensional incompressible two-fluid flows.

Chapter 3 is devoted to the spatial discretization, and the discontinuous Galerkin (DG) method. The DG method is based on spectral techniques. We shall therefore first describe a classical spectral method, introducing interpolating approximations, polynomial expansion and differentiation matrices. Then we shall introduce the DG method in one dimension, before we describe in details how to use the DG method in 2D for conservations laws. Finally we will present how to apply the DG method to an elliptic problem and the Navier-Stokes equations.

In Chapter 4 we discuss how to integrate differential equations forward in time, to achieve high temporal order of accuracy. We first present a very flexible high order method for solving ordinary differential equations (ODE). It is based on low order methods, and is almost as flexible as the low order method, while in the end still achieving high order. Then we will give an overview of theory from the solution of Differential Algebraic Equations (DAE): The incompressible Navier Stokes equations are difficult to solve due its combination of a differential equation and an algebraic mass conservation equation. We will in the overview give attention to the form of the incompressible Navier Stokes equations. Finally we shall combine the two, and present how to achieve high temporal order for the incompressible Navier Stokes, and outline the problems encountered.

The last of the three main parts, Chapter 5, describes in detail the free surface modeling and the level set method. This includes the properties which we require the level set to possess, and the processes needed to keep the properties. We will describe the reinitialization of the level set, how to apply boundary conditions and how to stabilize the reinitialization process. We shall furthermore compare the level set with other approaches and outline its strengths and weaknesses.

In Chapter 6 we present a few examples and numerical validations while Chapter 7 concludes with a few remarks and suggestions to further work.

CHAPTER 2

The Two-Fluid Navier Stokes Equations

The defining property of fluids, embracing both liquids and gases, lies in the ease with which they may be deformed.

— G. K. Batchelor, [7]

For a single fluid, the incompressible Navier Stokes equations are given by

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + f\ , \tag{2.1a}$$

$$\nabla\cdot\mathbf{u} = 0\ , \tag{2.1b}$$

where $\mathbf{u} = (u_1, ..., u_d)$ is the velocity vector, u_i is the ith cartesian component, p is the pressure, μ dynamic viscosity, ρ constant density, f an external force as e.g. gravity, and d is the dimension of the domain, usually two or three. The unknowns are the velocity vector $\mathbf{v}$ and the pressure p. The first (2.1a) is Newton's second law of motion $ma = f$ for a fluid, while the second is a mass and area conservation constraint, specifying that mass and area cannot shrink or disappear. For a derivation of the equations, see e.g. [2, 7, 22].

We shall consider the dynamics of two incompressible immiscible fluids, having different fluid properties. We will have water and air in our mind. Let the domain of water be denoted Ω_l and the domain of air Ω_g, where l and g denote liquid and gas respectively. The dynamics of each of the fluids are described by a set of Navier-Stokes equations,

$$\forall \mathbf{x} \in \Omega_l \ : \ \rho_l \left(\frac{\partial \mathbf{u}_l}{\partial t} + (\mathbf{u}_l \cdot \nabla)\mathbf{u}_l \right) = -\nabla p_l + \mu_l \nabla^2 \mathbf{u}_l \ \ , \tag{2.2a}$$

$$\nabla \cdot \mathbf{u}_l = 0 \ \ , \tag{2.2b}$$

$$\forall \mathbf{x} \in \Omega_g \ : \ \rho_g \left(\frac{\partial \mathbf{u}_g}{\partial t} + (\mathbf{u}_g \cdot \nabla)\mathbf{u}_g \right) = -\nabla p_g + \mu_g \nabla^2 \mathbf{u}_g \ \ , \tag{2.3a}$$

$$\nabla \cdot \mathbf{u}_g = 0 \ \ . \tag{2.3b}$$

In both fluids, ρ and μ represent the constant density and dynamic viscosity, respectively. In each fluid we also have the velocity field, $\mathbf{u}$, and the pressure field p.

The full computational domain consist of both fluids, $\Omega = \Omega_l \cup \Omega_g$. We shall assume Ω fixed in time, while both Ω_l and Ω_g are time dependent. We shall call the boundary of Ω, the global boundary, for $\partial\Omega$ while $\Gamma = \Omega_l \cap \Omega_g$ represents the interface between the two fluids. We will not make any assumption on the connectivity of Ω_l and Ω_g, i.e. both domains may consist of several smaller unconnected parts. Think of two droplets falling through the air. We shall assume the full computational domain Ω to be connected.

The interface between the two fluids can be geometrically very complex. It may consist of several disconnected parts, which can merge and split, as, e.g., when the two droplets collide. The interface is in mechanical equilibrium, meaning in terms of computational fluid dynamics (CFD) that the stress from each fluid balances on the interface, which can be written as

$$2\left(\mu_l \mathcal{D}_l - \mu_g \mathcal{D}_g\right) \cdot \mathbf{n}_\Gamma = \left(p_l - p_g\right) \mathbf{n}_\Gamma \ \ . \tag{2.4}$$

where $\mathcal{D}_l$ is the rate of deformation tensor, $\mathcal{D}_l = \frac{1}{2}(\nabla \mathbf{u}_l + (\nabla \mathbf{u}_l)^T)$ and similarly for $\mathcal{D}_g$, and $\mathbf{n}_\Gamma$ is the normal along Γ. Due to continuity in the velocities at the interface, $\mathcal{D}_l = \mathcal{D}_g$. If including surface tension, then the difference in stress will balance the surface tension force, i.e.

$$2\left(\mu_l \mathcal{D} - \mu_g \mathcal{D}\right) \cdot \mathbf{n}_\Gamma = \left(p_l - p_g + \sigma\kappa\right) \mathbf{n}_\Gamma \ \ , \tag{2.5}$$

where κ is the local curvature of the interface, $\kappa = \nabla \cdot \mathbf{n}_\Gamma$, and σ is the coefficient of surface tension, [17, 63].

To represent the interface and the two fluid domains, let us introduce the scalar level set function, ϕ, fulfilling

$$\phi(\mathbf{x}, t) = \begin{cases} > 0 & \mathbf{x} \in \Omega_l \\ 0 & \mathbf{x} \in \Gamma \\ < 0 & \mathbf{x} \in \Omega_g \end{cases} \ \ . \tag{2.6}$$

We assume ϕ to be continuous. We will define the interface as the zero contour of a level set function, and each of the fluids are defined by the sign of the level set function. From the level set function, we can derive the important interface properties as interface unit normal

$$\mathbf{n}_\Gamma = \left.\frac{\nabla\phi}{|\nabla\phi|}\right|_{\phi=0} \quad , \tag{2.7}$$

and interface curvature

$$\kappa = \nabla\cdot\mathbf{n}_\Gamma = \nabla\cdot\left.\frac{\nabla\phi}{|\nabla\phi|}\right|_{\phi=0} \quad . \tag{2.8}$$

If we define the global quantities

$$\mathbf{u} = \begin{cases} \mathbf{u}_l, & \mathbf{x}\in\Omega_l \\ \mathbf{u}_g, & \mathbf{x}\in\Omega_g \end{cases} \quad , \tag{2.9}$$

and likewise for the pressure, p, we can define global versions of the fluid constants based on the level set function,

$$\rho(\phi) = \rho_g + (\rho_l - \rho_g)H(\phi) \quad ,$$
$$\mu(\phi) = \mu_g + (\mu_l - \mu_g)H(\phi) \quad ,$$

both defined in the entire global domain Ω, $H(\phi)$ being the classical Heaviside function. Following [63], we can also combine the two sets of Navier Stokes equations into one and we arrive at a formulation

$$\begin{aligned} \rho(\phi)\left(\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) &= -\nabla p + \mu(\phi)\nabla\cdot\nabla\mathbf{u} - \sigma\delta(\phi)\kappa\mathbf{n}_\Gamma \quad , \\ \nabla\cdot\mathbf{u} &= 0 \quad , \end{aligned} \tag{2.10}$$

where $\delta(\phi) = \frac{\partial}{\partial\phi}H(\phi)$ is the Dirac delta function. Notice that $\delta(\phi)$ gives a contribution only at the interface, hence applying the surface tension force at the interface only. If we further seek a non-dimensional form, i.e., scaling each variable with a characteristic unit, using

$$\mathbf{x} = L\tilde{\mathbf{x}} \;\; , \;\; \mathbf{u} = U\tilde{\mathbf{u}} \;\; , \;\; t = (L/U)\tilde{t} \;\; , \;\; p = \rho_l U^2\tilde{p} \;\; , \;\; \rho = \rho_l\tilde{\rho} \;\; , \;\; \mu = \mu_l\tilde{\mu} \;\; , \tag{2.11}$$

where L is a characteristic length and U is a characteristic velocity, we get the dimensionless $\tilde{}$-variables. With these we recover the general form

$$\begin{aligned} \rho(\phi)\left(\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) &= -\nabla p + \frac{1}{Re}\mu(\phi)\nabla\cdot\nabla\mathbf{u} - \frac{1}{We}\delta(\phi)\kappa(\phi)\frac{\nabla\phi}{|\nabla\phi|} \quad , \\ \nabla\cdot\mathbf{u} &= 0 \quad , \end{aligned} \tag{2.12}$$

In this general form, which we shall consider subsequently, we now have

$$Re = \frac{\rho_l LU}{\mu_l} \quad , \tag{2.13}$$

$$We = \frac{\rho_l LU^2}{\sigma} \quad , \tag{2.14}$$

the Reynolds and Weber number respectively, as measures of the dynamics of the equations. The Reynolds number gives the ratio between inertia and viscous forces, while the Weber number compares inertia with surface tension forces.

According to the kinematic free surfaces condition, a condition that must be fulfilled at the interface, a particle on the surface will stay on the surface for the life of the surface. Thus, the surface follows the flow of the fluid. We shall update the level set accordingly. Assuming ϕ is differentiable, we will move ϕ using

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi = 0. \tag{2.15}$$

This linear convection equation, which we shall call the *level set equation*, moves the values of ϕ with the flow $\mathbf{u}$. It especially moves $\phi = 0$ in the direction of the flow at the interface, implying implicitly that also the surface moves with the flow.

We shall generally assume that $\partial\Omega = \partial\Omega_W \cup \partial\Omega_O$ where $\partial\Omega_W$ refers to hard walls where we impose a no-slip condition

$$\mathbf{u} = 0 \ \ , \ \ \mathbf{x} \in \partial\Omega_W \ \ ,$$

while we, at open boundaries, $\partial\Omega_O$, shall impose

$$(\mathbf{n} \cdot \nabla)\mathbf{u} = 0 \ \ , \ \ \mathbf{x} \in \partial\Omega_O \ \ .$$

CHAPTER 3

Spatial Scheme - Discontinuous Galerkin

All the mathematical sciences are founded on relations between physical laws and laws of numbers, so that the aim of exact science is to reduce the problems of nature to the determination of quantities by operations with numbers.

— James Clerk Maxwell,
on Faraday's Lines of Force (1856)

This chapter is devoted to the spatial scheme, i.e., how to approximate the spatial operators. We shall use the discontinuous Galerkin (DG) method, which was first presented in [58] in 1973. It is based on techniques from finite element and finite volume methods. We shall use a version utilizing high order elements. In that sense, it belongs to the family of spectral element methods, which is the most recent of the techniques for solving PDEs, including finite differences, finite volume, and finite element methods.

The advantage of spectral methods are their accuracy, as we shall present, providing spectral convergence[1] for smooth problems. Furthermore recent work has combined the high order of accuracy with geometric flexibility, see e.g. [34, 37].

In this chapter we will first describe the classical global spectral method. This will introduce the basics of spectral methods, polynomial expansion, node distribution, spectral integration and differentiation matrices, which we shall need in the next part: There we describe the discontinuous Galerkin in one of its most simple forms, in 1D for a conservation law. The following part extends the DG method to two dimensions and will provide details on how to efficiently and accurately compute and apply the spatial operators. Finally we shall apply the DG method on the incompressible Navier Stokes equations.

[1] Faster than any polynomial convergence.

3.1 Classical Spectral Method

This section on classical spectral method is inspired by the presentation in [70], extracting selected theory and properties needed in the following sections.

The idea behind this kind of numerical method is to approximate a continuous function $u(x)$, $x \in \Omega$, using a number of global basis functions defined on Ω.

For now, let the domain Ω be the real line, $\Omega : x \in [0; 1]$. Consider a grid having $N+1$ nodes, $\{x_0, x_1, ..., x_N\}$ as in Figure 3.1, and a set of $N+1$ basis-functions

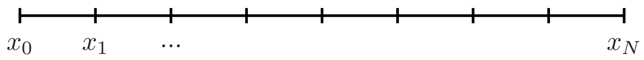


Figure 3.1: Grid having $N+1$ nodes

$\phi_i(x)$, we then define the interpolating approximation of $u(x)$

$$\tilde{u}(x) = \mathcal{I}_N u(x) = \sum_{i=0}^{N} \hat{u}_i \phi_i(x) \,. \tag{3.1}$$

Here $\mathcal{I}_N$ is the interpolation operator which interpolates the u-values between each grid point u_i, projecting u to the set of basis-functions. The coefficients $\hat{u}_i$ are defined by requiring the approximation $\tilde{u}$ to be exact at the nodes x_i. $\tilde{u}(x_i) = u(x_i)$, and they are in general determined as a solution to the linear system

$$\begin{bmatrix} \phi_0(x_0) & \dots & \phi_N(x_0) \\ \vdots & & \vdots \\ \phi_0(x_N) & \dots & \phi_N(x_N) \end{bmatrix} \begin{bmatrix} \hat{u}_0 \\ \vdots \\ \hat{u}_N \end{bmatrix} = \begin{bmatrix} u(x_0) \\ \vdots \\ u(x_N) \end{bmatrix} , \tag{3.2}$$

or in short $\mathsf{V}\hat{\mathbf{u}} = \mathbf{u}$. When $\phi_i(x_j) = x_j^i$ then V is the classical Vandermonde matrix [75]. Generalizing this, we will also call a matrix with elements $(\mathsf{V})_{ji} = \psi_i(x_j)$ a Vandermonde matrix, since they share some properties. Notice that the Vandermonde matrix V depends on the basis functions ϕ_j as well as the nodes x_i.

The choice of grid and basis functions is of great importance for the performance of the method, and it will depend on the problem to be solved. For example, for periodic problems, a common choice is trigonometric basis functions and an equidistant grid, and we would determine $\hat{u}_i$ by a Fourier Transform. However, for non-periodic problems, one would use polynomial basis functions and an unevenly spaced grid, examples are the Legendre grid, and as we shall meet later, the Chebyshev grid.

Assume that for a given grid we choose a "good" set of basis function, good in the sense that $\tilde{u}(x)$ is sufficiently close to $u(x)$, i.e. $||u(x) - \tilde{u}(x)||$ in an appropriate norm is small. Then we can operate on $\tilde{u}(x)$ instead of $u(x)$, and approximate e.g. the derivative by

$$w(x) = \frac{du(x)}{dx} \approx \frac{d\tilde{u}(x)}{dx} = \sum_{i=0}^{N} \hat{u}_i \frac{d\phi_i(x)}{dx} \,. \tag{3.3}$$

It is usually possible to expand the derivative of the basis-functions $\frac{d}{dx}\phi_i(x)$ in the $\phi_j(x)$ basis functions themselves, hence

$$\frac{d\phi_i(x)}{dx} = \sum_{j=0}^{N} d_{ji}\phi_j(x)\,. \tag{3.4}$$

Now the derivative (3.3) becomes

$$w(x) \approx \sum_{i=0}^{N}\sum_{j=0}^{N} d_{ji}\hat{u}_i\phi_j(x) = \sum_{j=0}^{N} \hat{w}_j\phi_j(x)\,. \tag{3.5}$$

Using the vectors $\hat{\mathbf{u}} = (\hat{u}_0, ..., \hat{u}_N)$ and $\hat{\mathbf{w}} = (\hat{w}_0, ..., \hat{w}_N)$, then to calculate the derivative (3.5) is nothing more than a matrix-vector product

$$\hat{\mathbf{w}} = \mathsf{D}\hat{\mathbf{u}}\,, \tag{3.6}$$

where D is the differentiation matrix and $(\mathsf{D})_{ji} = d_{ji}$.

The interpolating approximation and the differentiation matrix depend on the grid and the basis functions. We will here only consider non-periodic grids, since that is the most general and what we will be using later on. We shall use a common and simple choice for the basis functions, such that the interpolating approximation is the Lagrange interpolating polynomial. The Lagrange interpolating polynomial is the polynomial of degree N which passes through the function value at the $N+1$ nodes, $u_0, u_1, ..., u_N$. It is given by

$$\tilde{u}(x) = \mathcal{I}_N u(x) = \sum_{i=0}^{N} u(x_i) l_i, \qquad l_i = \prod_{\substack{k=0\\k\neq i}}^{N} \frac{x - x_k}{x_i - x_k}\,. \tag{3.7}$$

The polynomials l_i has the property that $l_i(x_j) = \delta_{ij}$, where δ_{ij} is the *Kronecker delta function*, hence the Vandermonde matrix is the identity matrix and $\hat{u}_i = u_i = u(x_i)$.

The definition above defines the basis from the grid, hence for a given grid, the basis is set. Next step is to determine a grid. The accuracy of the interpolating approximation $\tilde{u}$ depends strongly on the grid. Figure 3.2 shows a good and a bad example of an approximation, using an equidistant grid and a Chebyshev grid respectively. The example is especially noteworthy, since the most obvious grid, the equidistant grid, does not work. Often the *Chebyshev nodes*[2] are chosen, which is a set of nodes, including the end points of the interval and defined as

$$x_j = -\cos(j\pi/N), \quad j = 0, 1, ..., N\,. \tag{3.8}$$

The Chebyshev nodes are the projection of equally spaced nodes on the unit circle onto the x-axis, as in Figure 3.3. Chebyshev nodes are clustered more densely at

[2] Also called *Chebyshev-Lobatto nodes* or *Chebyshev-Gauss-Lobatto nodes*.

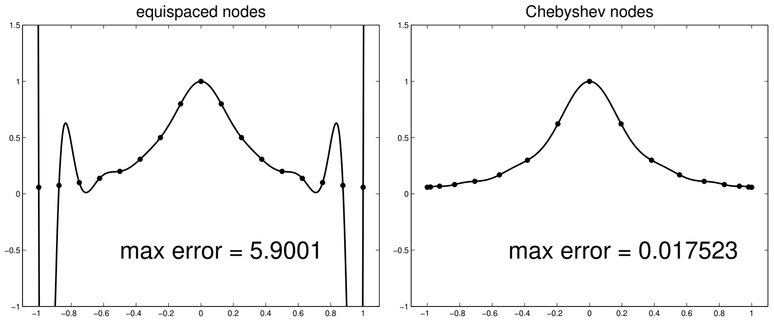


Figure 3.2: the interpolating $\tilde{u}$ of $u(x) = \frac{1}{1+16x^2}$ on an equidistant grid (left) and a Chebyshev grid (right) - example taken from [70]

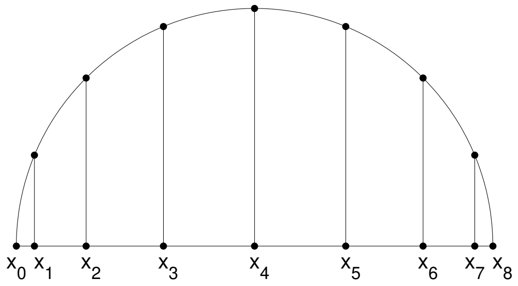


Figure 3.3: Chebyshev grid with $N = 9$ nodes

the end of the interval. The distance between adjacent nodes at the end is of order $\mathcal{O}\left(N^{-2}\right)$, while in the middle the distance is of order $\mathcal{O}\left(N^{-1}\right)$. There are other well working grids for the polynomial approximation, as e.g. the Legendre nodes, which are the nodes for the Gaussian quadrature. However, all grids share the property of being more densely distributed at the interval ends.

A Lagrange interpolating polynomial based on the Chebyshev nodes will have spectral accuracy, meaning that for sufficiently smooth functions, $\tilde{u}$ will converge towards u faster than Δx^m for any $m > 0$, i.e. the convergence is faster than any polynomial. See e.g. [70] for details.

The differentiation matrix D for the Chebyshev grid can be found analytically, see e.g. [70]. The resulting derivative has also spectral accuracy. Differentiation matrices for spectral methods are full matrices. Hence the price of calculating the derivative is $\mathcal{O}\left(N^2\right)$, whereas a finite difference/finite element is only $\mathcal{O}(N)$.

3.1.1 Boundary Value Problems

We shall now consider how to solve boundary value problems and especially how to impose boundary conditions for the classical spectral method. Consider e.g.

$$u_{xx} = f, \qquad u(-1) = u_a, \quad u(1) = u_b\,. \tag{3.9}$$

We can approximate the second derivative by the square of the differentiation matrix, hence

$$\mathsf{D}^2\mathbf{u} = \mathbf{f}\,. \tag{3.10}$$

This is a system of $N+1$ equations, one for each of the $N+1$ unknowns in $\mathbf{u}$, including the interval endpoints. Boundary conditions are not yet imposed, hence including the boundary equations we have $N+3$ equations, therefore we must drop 2 equations. There are traditionally two ways of imposing the boundary conditions

- By restricting the solution space to satisfy the boundary conditions
- By adding equations that enforce the boundary conditions

The first approach works for homogeneous boundary conditions, $u_a = u_b = 0$, by requiring that all basis functions are zero on the boundary. That is the case when removing the basis functions based on each of the boundary nodes, l_0 and l_N. This can be accomplished from Equation (3.10) by removing the first and last element of the solution vector $\mathbf{u}$ and right hand side $\mathbf{f}$, and removing the first and last column and row from D

$$\sum_{j=1}^{N-1} (\mathsf{D}^2)_{ij} u_j = f_i, \quad i = 1, ..., N-1\,. \tag{3.11}$$

Notice the numbering, $i, j = 1, ..., N-1$, whereas the matrix equation (3.10) can be written in the same manner, just changing $i, j = 0, ..., N$. This may be written more conveniently in matrix form using a Matlab type notation[3]

$$\mathsf{D}_{(1:N-1,1:N-1)}\; u_{(1:N-1)} = f_{(1:N-1)}\,. \tag{3.12}$$

For general inhomogeneous Dirichlet boundary conditions we can add a contribution on the right hand side coming from the endpoints

$$\sum_{j=1}^{N-1} (\mathsf{D}^2)_{ij} u_j = f_i - (\mathsf{D}^2)_{i0} u_a - (\mathsf{D}^2)_{iN} u_b, \quad i = 1, ..., N-1\,. \tag{3.13}$$

The linear system is in both cases of size $(N-1) \times (N-1)$.

[3] Indices here start at 0, while Matlab indices start at 1, hence we have removed row and column with index 0 and N

In the second approach we would replace the first and last equation with an equation satisfying the boundary conditions,

$$\sum_{j=0}^{N} (\mathsf{D}^2)_{ij} u_j = f_i \quad i = 1, ..., N-1, \tag{3.14a}$$

$$u_0 = u_a, \quad u_N = u_b\,, \tag{3.14b}$$

giving a linear system of size $(N+1) \times (N+1)$. From this system we can get Equation (3.13) by eliminating u_0 by u_a and u_N by u_b and moving terms involving u_a and u_b to the right hand side. The difference between the two approaches is basically that u_0 and u_N is part of the solution vector in the latter approach, but not in the former.

The first approach is in general impossible if the boundary conditions become more complex, e.g. if having a Neumann condition on the left endpoint, $u_x(-1) = g$. The second approach is, however, fairly straight forward, replacing the first equation with an equation for the first derivative

$$\sum_{j=0}^{N} (\mathsf{D}^2)_{ij} u_j = f_j \quad i = 1, ..., N-1, \tag{3.15a}$$

$$\sum_{j=0}^{N} (\mathsf{D})_{0j} u_j = g, \qquad u_N = u_b\,. \tag{3.15b}$$

This again forms a linear system of size $(N+1) \times (N+1)$, which we need to solve to obtain the solution.

A note: For the second approach, it is not vital which of the equations that are replaced by the boundary equations, e.g. would

$$\sum_{j=0}^{N} (\mathsf{D}^2)_{ij} u_j = f_j \quad i \in \{0, ..., N\} \setminus p, q, \tag{3.16a}$$

$$\sum_{j=0}^{N} (\mathsf{D})_{0j} u_j = g, \qquad u_N = u_b\,. \tag{3.16b}$$

give a solution for all choices of p, q. The i defines on which nodes we want to approximate the equation, and as long as it is done on $N-1$ distinct nodes within the interval, we get a solution. However, from a numerical point of view, removing the first and last equation will give the most well conditioned system and the most accurate result.

3.1.2 Multi Domain Spectral Methods

Spectral methods have the benefit of high order and accuracy for smooth problems. However, it does not work well for problems where the solution is less smooth or

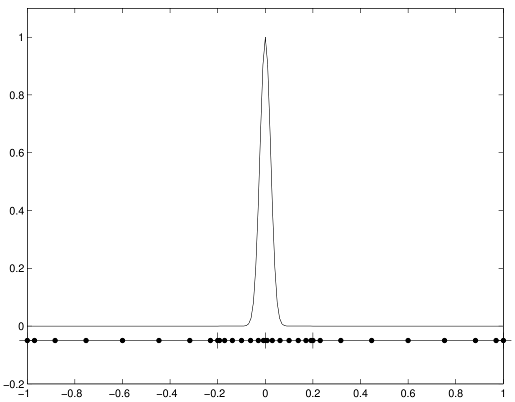


Figure 3.4: Localized solution

when the solution is localized in space, i.e. changes a lot within only a small part of the domain, as in Figure 3.4. To approximate this kind of solution well, we would need a very high order spectral method, which spreads a lot of nodes over the entire interval to resolve the solution. On a Chebyshev grid, most nodes are situated close to the interval endpoints where in this case the solution is basically constant. Hence on most of the interval we could approximate the solution better using a lower order method, and less number of nodes.

Instead we can split the interval into several smaller non-overlapping intervals and apply a spectral method on each part. In this way we can obtain the accuracy were we need it and reduce the size of the system.

On each interval we apply a spectral method, hence we need to apply boundary conditions on each interval boundary. The global boundaries are treated as before, but we must supply boundary conditions for the inner boundaries. For many problems it would suffice to require that the right value on interval 1, u_N^1, equals the left value on interval 2, u_0^2, hence we can eliminate one of them from the system. For systems including higher derivatives, this becomes more complicated, and is not straight forward.

3.2 Discontinuous Galerkin

We shall describe the discontinuous Galerkin (DG) method first in 1D, following to some extent the presentation in [62]. We shall later extend it to higher dimension. Consider the conservation law

$$\frac{\partial}{\partial t}q - \frac{\partial}{\partial x}\mathbf{f}(q) = 0, \quad q \in [0;1], \quad q(0) = g, \quad q(1) = h\,. \tag{3.17}$$

We split the interval $\mathcal{D} = [0;1]$ into K non-overlapping intervals, $\bigcup_k \mathcal{D}^k = \mathcal{D}$. The interval sizes may vary as desired, as depicted in Figure 3.5. Within each interval

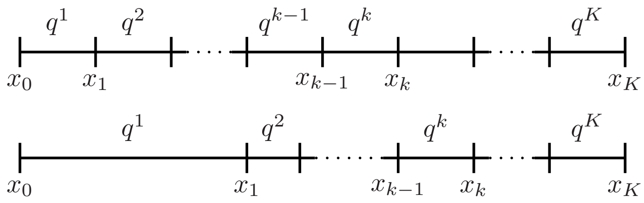


Figure 3.5: Grid for the DG method in 1D

$\mathcal{D}_k$ we wish to apply a spectral polynomial method. First we define a grid within each interval having $N+1$ nodes,

$$x_{k-1} = x_0^k < x_1^k < ... < x_N^k = x_k \tag{3.18}$$

and then we approximate using Lagrange interpolating polynomials

$$\tilde{q}^k(x) = \mathcal{I}_N q^k(x) = \sum_{i=0}^{N} \hat{q}_i^k \; l_i(x), \tag{3.19}$$

$$\tilde{f}^k(q(x)) = \mathcal{I}_N f(q^k(x)) = \sum_{i=0}^{N} \hat{f}_i^k \; l_i(x) \; , \tag{3.20}$$

i.e., the space in which we search our solution is spanned by $\{l_i\}$. We will use the notation

$$\tilde{f}_l^k = \tilde{f}^k(q(x_l^k)), \tag{3.21}$$

hence superscript k denotes the interval and subscript l denotes the local node number in the interval. In each interval we wish to satisfy Equation (3.17) in a weak Galerkin sense. We multiply by a test-function v and integrate over the interval

$$\int_{\mathsf{I}^k} \left(\frac{\partial \tilde{q}^k}{\partial t} - \frac{\partial \tilde{f}^k}{\partial x} \right) v \; \mathrm{d}x = 0 \tag{3.22}$$

which should be valid for any function v in some test space. Integrating by parts we get

$$\int_{\mathsf{I}^k} \frac{\partial \tilde{q}^k}{\partial t} v \; \mathrm{d}x + \int_{\mathsf{I}^k} \tilde{f}^k \frac{\partial v}{\partial x} \; \mathrm{d}x - \tilde{f}_N^k \; v(x_N^k) + \tilde{f}_0^k \; v(x_0^k) = 0 \tag{3.23}$$

which is the basis for finite element methods and also discontinuous Galerkin methods . Notice that $\tilde{f}_N^k$ simply refers to f at the right endpoint in the interval k, $\tilde{f}_N^k = \tilde{f}^k(q(x_N^k)) = \tilde{f}^k(q(x_k))$. Finite element methods uses test functions which satisfy a homogeneous Dirichlet condition on the interval boundary

$$v(x_0^k) = v(x_N^k) = 0 \tag{3.24}$$

thereby removing the two rightmost terms before the equality sign. The DG method uses another approach, and uses a test function space spanned by the basis for the Lagrange interpolating polynomials l_j, hence

$$\int_{\mathsf{I}^k} \frac{\partial \tilde{q}^k}{\partial t} l_j \,\mathrm{d}x + \int_{\mathsf{I}^k} \tilde{f}^k \frac{\partial l_j}{\partial x} \,\mathrm{d}x - \tilde{f}_N^k \; l_j(x_N^k) + \tilde{f}_0^k \; l_j(x_0^k) = 0, \tag{3.25}$$

which must be valid for all the basis functions l_j. This introduces an ambiguity in the equation, since now both the solution q, and hence the right hand side f, and also the test functions l_j are discontinuous at the interval endpoints x_k and x_{k-1}. This means that the values $\tilde{f}_N^k \; l_j(x_N^k)$ and $\tilde{f}_0^{k+1} \; l_j(x_0^{k+1})$ refers to the same point in space, x_k, but will not have the same value. A typical example is shown in

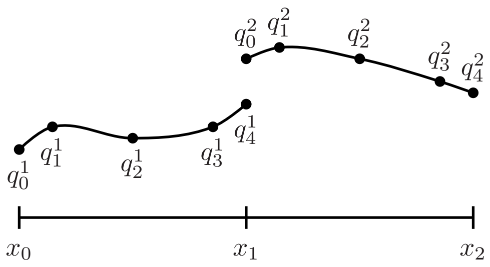


Figure 3.6: Discontinuous over interval boundaries

Figure 3.6. This is not necessarily a problem, on contrary it opens the opportunity to design these terms for accuracy and stability.

Example 3.1: Constant polynomial approximation
Consider the linear transport equation, where $f(q) = q$

$$\frac{\partial q}{\partial t} - \frac{\partial q}{\partial x} = 0, \quad q \in [0;1], \quad q(1) = g(t). \tag{3.26}$$

where initial and boundary data $g(t)$ will move to the left with speed one.

We split the interval into K subintervals, and put only one grid point within each interval , hence the Lagrange interpolating polynomial will be the constant function within each interval, $l_0 = 1$. Now replace $\tilde{f}_N^k$ and $\tilde{f}_0^k$ in Equation (3.25) by a *numerical flux*

$$\tilde{f}_N^k \rightarrow \quad f_k^* \;= f_k^*(q^k(x_k), q^{k+1}(x_k)) \tag{3.27a}$$
$$\tilde{f}_0^k \rightarrow f_{k-1}^* = f_{k-1}^*(q^{k-1}(x_{k-1}), q^k(x_{k-1})) \tag{3.27b}$$

The numerical flux basically decides which value of q to use on the interval endpoints where $q^k(x_k)$ and $q^{k+1}(x_k)$ refers to the same point in space, the ambiguity. The numerical flux is used for imposing boundary conditions on each interval, thereby specifying how information passes between adjacent intervals. In this case we will use up-winding fluxes, i.e. using the value in the direction where information is coming from, in this case the rightmost interval

$$f_k^* = f(q^{k+1}(x_k)) = q^{k+1} \tag{3.28a}$$
$$f_{k-1}^* = f(q^k(x_{k-1})) = q^k \tag{3.28b}$$

Since l is constant, $\frac{\partial l}{\partial x} = 0$, defining $\Delta x_k = (x_k - x_{k-1})$, Equation (3.25) becomes

$$\Delta x_k \frac{\partial q^k}{\partial t} - q^{k+1} + q^k = 0 \tag{3.29}$$

or

$$\frac{\partial q^k}{\partial t} - \frac{1}{\Delta x_k}(q^{k+1} - q^k) = 0 \tag{3.30}$$

giving the same scheme as if using a first order up-winding finite volume method, and also finite difference for equidistant grids.

End of Example 3.1.

Now return to Equation (3.25), using numerical fluxes as in Example 3.1, we have

$$\int_{\mathsf{I}^k} \frac{\partial \tilde{q}^k}{\partial t} l_j \, \mathrm{d}x + \int_{\mathsf{I}^k} \tilde{f}^k \frac{\partial l_j}{\partial x} \, \mathrm{d}x - f_k^* \, l_j(x_N^k) + f_{k-1}^* \, l_j(x_0^k) = 0. \tag{3.31}$$

Performing yet another integration by parts, we get what is often called the strong formulation of the DG method

$$\int_{\mathsf{I}^k} \left(\frac{\partial \tilde{q}^k}{\partial t} - \frac{\partial \tilde{f}^k}{\partial x} \right) l_j \, \mathrm{d}x - \left(f_k^* - \tilde{f}_N^k \right) \, l_j(x_N^k) + \left(f_{k-1}^* - \tilde{f}_0^k \right) \, l_j(x_0^k) = 0, \tag{3.32}$$

which still must hold for all l_j. Consider the integral part, and insert the polynomial approximation formulas (3.19) and (3.20)

$$\int_{\mathsf{I}^k} \frac{\partial}{\partial t} \left(\sum_{i=0}^{N} \hat{q}_i^k \, l_i \right) \, l_j - \frac{\partial}{\partial x} \left(\sum_{i=0}^{N} \hat{f}_i^k \, l_i \right) \, l_j \, \mathrm{d}x \tag{3.33a}$$
$$= \sum_{i=0}^{N} \frac{\partial \hat{q}_i^k}{\partial t} \int_{\mathsf{I}^k} l_i \, l_j \, \mathrm{d}x - \sum_{i=0}^{N} \hat{f}_i^k \int_{\mathsf{I}^k} \frac{\partial l_i}{\partial x} \, l_j \, \mathrm{d}x, \quad \forall j. \tag{3.33b}$$

Defining the operators M and S as having elements

$$(\mathsf{M})_{ji} = \int_{\mathsf{I}^k} l_i \, l_j \, \mathrm{d}x, \qquad (\mathsf{S})_{ji} = \int_{\mathsf{I}^k} \frac{\partial l_i}{\partial x} \, l_j \, \mathrm{d}x, \tag{3.34}$$

then all the j versions of Equation (3.32) can be written in one equation on matrix form as

$$\mathsf{M}\,\frac{\partial}{\partial t}\begin{pmatrix}\hat{q}_0^k\\ \hat{q}_1^k\\ \vdots\\ \hat{q}_{N-1}^k\\ \hat{q}_N^k\end{pmatrix} - \mathsf{S}\begin{pmatrix}\hat{f}_0^k\\ \hat{f}_1^k\\ \vdots\\ \hat{f}_{N-1}^k\\ \hat{f}_N^k\end{pmatrix} + \begin{pmatrix}f_{k-1}^* - \hat{f}_0^k\\ 0\\ \vdots\\ 0\\ -(f_k^* - \hat{f}_N^k)\end{pmatrix} = 0 \tag{3.35}$$

or in short

$$\mathsf{M}\,\frac{\partial \hat{\mathbf{q}}^k}{\partial t} - \mathsf{S}\,\hat{\mathbf{f}}^k - \left(f_k^* - \hat{f}_N^k\right)\mathbf{1}_N + \left(f_{k-1}^* - \hat{f}_0^k\right)\,\mathbf{1}_0 = 0. \tag{3.36}$$

where the vector $\mathbf{1}_p$ has elements $(\mathbf{1}_p)_j = \delta_{jp}$, i.e. value one at position p and zero elsewhere, coming from the fact that $l_j(x_p^k) = \delta_{jp}$. Similarly the weak version Equation (3.31)

$$\mathsf{M}\,\frac{\partial \hat{\mathbf{q}}^k}{\partial t} + \mathsf{S}^T\,\hat{\mathbf{f}}^k - f_k^*\mathbf{1}_N + f_{k-1}^*\mathbf{1}_0 = 0. \tag{3.37}$$

To solve the conservation law, we now only need to specify the numerical flux f_k^*.

Example 3.2: Linear transport equation
Using up-winding fluxes as in Example 3.1, i.e.

$$f_k^* = f(q^{k+1}(x_k)) = q_0^{k+1} \tag{3.38a}$$

$$f_{k-1}^* = f(q^k(x_{k-1})) = q_0^k \tag{3.38b}$$

we get the scheme

$$\mathsf{M}\,\frac{\partial \hat{\mathbf{q}}^k}{\partial t} - \mathsf{S}\,\hat{\mathbf{q}}^k - \left(q_0^{k+1} - q_N^k\right)\mathbf{1}_N + \left(q_0^k - q_0^k\right)\,\mathbf{1}_0 = 0. \tag{3.39a}$$

$$\mathsf{M}\,\frac{\partial \hat{\mathbf{q}}^k}{\partial t} - \mathsf{S}\,\hat{\mathbf{q}}^k - \left(q_0^{k+1} - q_N^k\right)\mathbf{1}_N = 0. \tag{3.39b}$$

or in the weak version

$$\mathsf{M}\,\frac{\partial \hat{\mathbf{q}}^k}{\partial t} + \mathsf{S}^T\,\hat{\mathbf{q}}^k - q_0^{k+1}\mathbf{1}_N + q_0^k\mathbf{1}_0 = 0. \tag{3.40}$$

End of Example 3.2.

3.3 Discontinuous Galerkin in 2D

Having developed the classical spectral method and the DG method in 1D, it is now time to consider 2D.

Both the classical spectral method and the DG method is fairly easily extended to rectangular domains, by applying the grids and operators in each dimension in a kind of Kronecker product approach. DG methods based on rectangular and, by a generalization, quadrilateral elements, are applicable to a great variety of domains, and has been extensively used. A very relevant example is [67], also considering the level equation.

We shall enlarge the variety of domains even further by using a fully unstructured triangular grid.

The DG methods on triangular elements for conservation laws is described partly in [32, 34, 35, 37]. This section collects selected parts of the four articles, giving a methodical presentation of how the DG method works, with focus on implementation details, and in the end describing a small implementation.

3.3.1 DG for Conservation Laws

We shall describe the DG method again first for the general conservation law

$$\frac{\partial}{\partial t} q + \nabla \cdot \mathbf{f}(q) = 0, \tag{3.41}$$

and thereafter extend it to the incompressible Navier Stokes. However we shall describe the general setup first.

We split our domain Ω into K non-overlapping triangular elements D^k, such that $\bigcup_k \mathsf{D}^k = \Omega$. Define a standard triangular element D,

$$\mathsf{D} = \{(r,s) \in \mathsf{R}^2 \mid r, s \geq -1 \, ; \, r + s \leq 0\}. \tag{3.42}$$

as depicted in Figure 3.7. For any given element D^k we define a smooth bijective

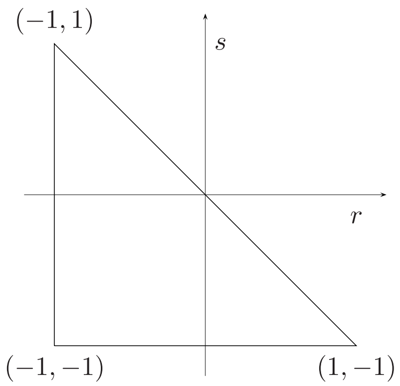


Figure 3.7: Standard triangular element

mapping to the standard element on the form

$$\begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} r_x & r_y \\ s_x & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \mathsf{J}\,\mathbf{x} + \mathbf{t}. \tag{3.43}$$

where the coefficients r_x, r_y, s_x, and s_y in the transformation Jacobian J may depend on space (x, y). This allows for a great variety of linear and also non-linear mappings, as long as they are smooth and bijectictive, hence allowing triangular elements with curvilinear sides still to be mapped onto the straight sided standard triangle D.

We shall only consider straight sided triangular elements D^k, since it does simplify the scheme and the presentation of it, e.g. the transformation Jacobian J becomes constant within each element D^k.

If we construct operators on the standard element, then operators on each element D^k are given by a linear combination of those on the standard element, e.g., for differentiation by application of the chain rule:

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial}{\partial s}\frac{\partial s}{\partial x} = r_x\frac{\partial}{\partial r} + s_x\frac{\partial}{\partial s} \ , \tag{3.44a}$$

$$\frac{\partial}{\partial y} = \frac{\partial}{\partial r}\frac{\partial r}{\partial y} + \frac{\partial}{\partial s}\frac{\partial s}{\partial y} = r_y\frac{\partial}{\partial r} + s_y\frac{\partial}{\partial s} \ . \tag{3.44b}$$

Utilizing the bijective mapping for each element, we can restrict attention to the standard triangular element D. As in the 1D case, we will represent the value of a function q in the standard element using a 2D Lagrange interpolating polynomial based on nodes $\mathbf{r}_i$

$$q(\mathbf{r}, t) \approx \tilde{q}(\mathbf{r}, t) = \sum_{i=0}^{N} \hat{q}_i(t) L_i(\mathbf{r}) \ , \tag{3.45}$$

where $N + 1$ is the number of nodes in the element, $L_i(\mathbf{r})$ is the ith nodal basis function based on node $\mathbf{r}_i$, and $\hat{q}_i(t) = q(\mathbf{r}_i, t)$ is the value at node $\mathbf{r}_i$. The nodal basis functions L_i has the property

$$L_i(\mathbf{r}_j) = \delta_{ij}. \tag{3.46}$$

and is the 2D equivalence of Equation (3.7). The nodal basis functions L_j does not in general have a simple closed formula but can be found as follows: We shall require the nodal basis functions to span a 2D polynomial space P^n of order n, i.e.

$$\mathsf{P}^n = \text{span}\{x^{\alpha_1} y^{\alpha_2}\}, \qquad \alpha_1, \alpha_2 \geq 0, \quad \alpha_1 + \alpha_2 \leq n \ . \tag{3.47}$$

The polynomial space P^n has dimension $(n+1)(n+2)/2$. A basis spanning P^n can be ordered and illustrated as

$$\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & x & & y & & & \\
 & & x^2 & & xy & & y^2 & & \\
 & x^3 & & x^2y & & xy^2 & & y^3 & \\
\dots & & \dots & & \dots & & \dots & & \dots
\end{array}$$

i.e. the Pascal triangle of order n. The task is to find a valid nodal distribution within the triangle, i.e. a distribution giving a set of basis functions $\{L_i\}$ which spans the entire P^n and has the same dimension. We will define what we require for a nodal distribution to be valid later.

Assume we have $(n+1)(n+2)/2 = N+1$ nodes in the standard element. Let p_m, $m = 0, ..., N$, be the mth basis function spanning P^n, e.g. the mth function in the Pascal triangle. Write the L_i in the p_j basis,

$$L_i = \sum_{m=0}^{N} a_{mi} p_m \; . \tag{3.48}$$

We now want to determine the coefficients a_{im} such that $L_i(\mathbf{r}_j) = \delta_{ij}$ for all $\mathbf{r}_j$,

$$\sum_{m=0}^{N} a_{mi} p_m(\mathbf{r}_j) = \delta_{ij} \; , \qquad \forall j \; , \tag{3.49}$$

which is found by solving the linear system

$$\begin{bmatrix} p_0(\mathbf{r}_0) & \dots & p_N(\mathbf{r}_0) \\ \vdots & & \vdots \\ p_0(\mathbf{r}_N) & \dots & p_N(\mathbf{r}_N) \end{bmatrix} \begin{bmatrix} a_{0i} \\ \vdots \\ a_{Ni} \end{bmatrix} = \begin{bmatrix} \delta_{0i} \\ \vdots \\ \delta_{Ni} \end{bmatrix} \tag{3.50}$$

or in short $\mathsf{V}\mathbf{a}_i = \mathbf{1}_i$. The matrix V is the 2-dimensional Vandermonde matrix. If we define the coefficient matrix A having elements $(\mathsf{A})_{ni} = a_{ni}$, then all the a_{ni} coefficient are found by

$$\mathsf{V}\,\mathsf{A} = \mathsf{I} \quad \Leftrightarrow \quad \mathsf{A} = \mathsf{V}^{-1}. \tag{3.51}$$

where I is the identity matrix.

Definition 3.3.1. We will call the nodal distribution *valid* when the Vandermonde matrix is regular, i.e. when V^{-1} exists.

Examples of valid and "almost optimal" nodal sets can be found in [32], Figure 3.8 illustrates 4 of these nodal sets on the standard element spanning the polynomial space P^n of order $n = 3$, 5, 7 and 9 respectively. Notice how the nodes are distributed more densely along the edges, and especially close to the corners. This is the 2D equivalent for solving the problems depicted in Figure 3.2. In Appendix A a number of the nodal basis functions L_i is depicted for the element of order 5 and 8.

Let us return to the conservation law, Equation (3.41). We will approximate our solution by Lagrange interpolating polynomials

$$q \approx \tilde{q} = \sum_{i=0}^{N} \hat{q}_i L_i(\mathbf{x}) \tag{3.52a}$$

$$\mathbf{f} \approx \tilde{\mathbf{f}} = \sum_{i=0}^{N} \hat{\mathbf{f}}_i L_i(\mathbf{x}) \tag{3.52b}$$

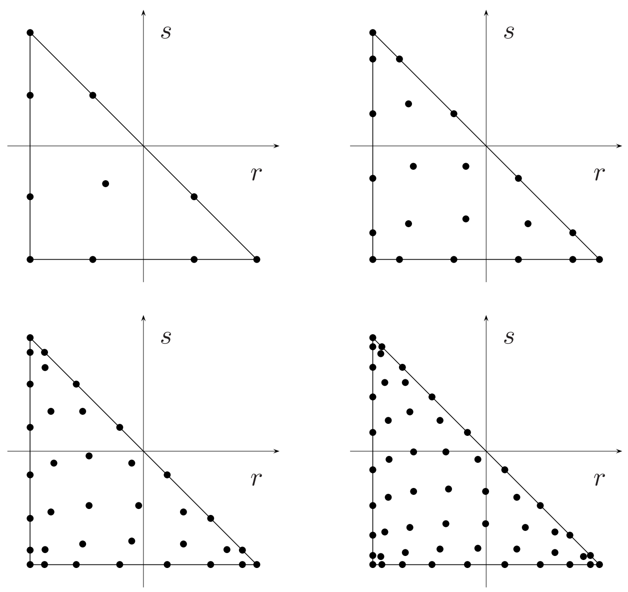


Figure 3.8: Nodes in standard element of order 3, 5, 7 and 9

Consider the weak element-wise formulation of the conservation law

$$\int_{\mathsf{D}} \left(\frac{\partial \tilde{q}}{\partial t} + \nabla \cdot \tilde{\mathbf{f}} \right) L_i(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 0, \quad \forall \, i. \tag{3.53}$$

where we assume that $\{L_i(\mathbf{x})\}$ spans the polynomial space P^n. Integration by parts yields

$$\int_{\mathsf{D}} \frac{\partial \tilde{q}}{\partial t} L_i(\mathbf{x}) - \tilde{\mathbf{f}} \cdot \nabla L_i(\mathbf{x}) \, \mathrm{d}\mathbf{x} = - \oint_{\partial \mathsf{D}} \mathbf{n} \cdot \mathbf{f}^* L_i(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \quad \forall \, i, \tag{3.54}$$

where $\mathbf{n}$ is the outward pointing normal vector on $\partial\mathsf{D}$. On the right hand side, the numerical flux $\mathbf{f}^*$ is introduced replacing $\tilde{\mathbf{f}}$, which is used for imposing boundary conditions on each element, thereby specifying how information passes between adjacent elements. This is the classical discontinuous Galerkin (DG) method in weak form.

Notice, as in the 1D case, both the solution $\tilde{q}$ and the test functions (the nodal basis functions L_i) are discontinuous at element boundaries, the ambiguity being resolved by the numerical flux.

Integrating by parts again, we end up with the strong formulation of the DG method,

$$\int_{\mathsf{D}} \left(\frac{\partial \tilde{q}}{\partial t} + \nabla \cdot \tilde{\mathbf{f}} \right) L_i(\mathbf{x}) \, \mathrm{d}\mathbf{x} = \oint_{\partial \mathsf{D}} \mathbf{n} \cdot (\tilde{\mathbf{f}} - \mathbf{f}^*) L_i(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \quad \forall \, i. \tag{3.55}$$

In what remains we shall focus on this form, using the weak form only for illustrative purposes. Inserting Equations (3.52) and after some rearrangement we get

$$\sum_{j=0}^{N} \frac{\partial \hat{q}_j}{\partial t} \int_{\mathsf{D}} L_j\, L_i \,\mathrm{d}\mathbf{x} + \sum_{j=0}^{N} \hat{\mathbf{f}}_j \cdot \int_{\mathsf{D}} \nabla L_j\, L_i \,\mathrm{d}\mathbf{x} \\ = \mathbf{n} \cdot \left(\sum_{j=0}^{N} \hat{\mathbf{f}}_j - \hat{\mathbf{f}}_j^* \right) \oint_{\partial \mathsf{D}} L_j L_i \,\mathrm{d}\mathbf{x}, \quad \forall\, i. \tag{3.56}$$

Introducing the vector $\hat{\mathbf{q}} = \{\hat{q}_1, ..., \hat{q}_N\}$ and similar for $\hat{\mathbf{f}}$, and defining the operators $\hat{\mathsf{M}}$, $\hat{\mathsf{S}}$ and $\hat{\mathsf{F}}$ having elements

$$(\hat{\mathsf{M}})_{ij} = \int_{\mathsf{D}} L_j L_i \,\mathrm{d}\mathbf{x}, \tag{3.57a}$$

$$(\hat{\mathsf{S}})_{ij} = \int_{\mathsf{D}} \nabla L_j L_i \,\mathrm{d}\mathbf{x}, \tag{3.57b}$$

$$(\hat{\mathsf{F}})_{ij} = \oint_{\partial \mathsf{D}} L_j L_i \,\mathrm{d}\mathbf{x}, \tag{3.57c}$$

the DG method in the weak and strong formulation translates into

$$\hat{\mathsf{M}} \frac{\partial \hat{\mathbf{q}}}{\partial t} - \hat{\mathsf{S}}^T \cdot \hat{\mathbf{f}} = -\hat{\mathsf{F}}(\mathbf{n} \cdot \hat{\mathbf{f}}^*) \tag{3.58}$$

$$\hat{\mathsf{M}} \frac{\partial \hat{\mathbf{q}}}{\partial t} + \hat{\mathsf{S}} \cdot \hat{\mathbf{f}} = \hat{\mathsf{F}}(\mathbf{n} \cdot (\hat{\mathbf{f}} - \hat{\mathbf{f}}^*)) \tag{3.59}$$

respectively.

A note on the notation: We are a little sloppy with the notation here, since $\mathbf{f}$ in general is multi-dimensional, e.g. in 2 dimensions $\mathbf{f} = (f_x, f_y)$. Hence the coefficient vector $\hat{\mathbf{f}}$ is actually denoting a vector of vectors, $\hat{\mathbf{f}} = (\hat{\mathbf{f}}_x, \hat{\mathbf{f}}_y)$, and similar is $\hat{\mathsf{S}}$ denoting a vector of stiffness matrices

$$\hat{\mathsf{S}} = (\hat{\mathsf{S}}_x, \hat{\mathsf{S}}_y)\,, \qquad \hat{\mathsf{S}}_x = \int_{\mathsf{D}} \frac{\partial}{\partial x} L_j L_i \,\mathrm{d}\mathbf{x}, \qquad \hat{\mathsf{S}}_y = \int_{\mathsf{D}} \frac{\partial}{\partial y} L_j L_i \,\mathrm{d}\mathbf{x}\,. \tag{3.60}$$

We define the inner product and transpose as

$$\hat{\mathsf{S}} \cdot \hat{\mathbf{f}} = \hat{\mathsf{S}}_x\, \hat{\mathbf{f}}_x + \hat{\mathsf{S}}_y\, \hat{\mathbf{f}}_y \tag{3.61a}$$

$$\hat{\mathsf{S}}^T \cdot \hat{\mathbf{f}} = \hat{\mathsf{S}}_x^T\, \hat{\mathbf{f}}_x + \hat{\mathsf{S}}_y^T\, \hat{\mathbf{f}}_y\,, \tag{3.61b}$$

and with this in mind, we can continue using the more compact notation.

The numerical flux $\hat{\mathbf{f}}^*$ is defined along the boundary edge of an element and is a function of q on the boundary edge of the local element $\mathbf{q}^-$ and the neighboring element $\mathbf{q}^+$. Three different fluxes are utilized here, a central flux, the Lax Friedrich flux and a flux for applying boundary conditions:

$$\hat{\mathbf{f}}^*_C(\mathbf{q}^-, \mathbf{q}^+) = \frac{1}{2}(\mathbf{f}(\mathbf{q}^-) + \mathbf{f}(\mathbf{q}^+)) \ , \tag{3.62a}$$

$$\hat{\mathbf{f}}^*_{LF}(\mathbf{q}^-, \mathbf{q}^+) = \frac{1}{2}(\mathbf{f}(\mathbf{q}^-) + \mathbf{f}(\mathbf{q}^+)) - \frac{c}{2}(\mathbf{q}^+ - \mathbf{q}^-) \ , \tag{3.62b}$$

$$\hat{\mathbf{f}}^*_{BC}(\mathbf{q}^-, \mathbf{q}^+) = \hat{\mathbf{f}}(\mathbf{q}_{BC}) \ , \tag{3.62c}$$

where c is the maximum wave-speed, $c = \max|\lambda(\frac{\partial \mathbf{f}}{\partial \mathbf{q}})|$. We will in general use the central flux for linear $\mathbf{f}$ and a Lax-Friedrich flux for nonlinear $\mathbf{f}$,

The conditioning of the operators $\hat{\mathsf{M}}$, $\hat{\mathsf{S}}$ and $\hat{\mathsf{F}}$ depend strongly on the distribution of nodes within the standard element. The distribution sets used here are shown in [32] to produce well conditioned operator spanning polynomial spaces up till 16th order, P_{16}. The nodal distribution takes into account that a boundary integral is a vital part of the method, hence nodes are placed on the element boundary such that the boundary integral can be easily evaluated, using Gauss-Lobatto nodes.

3.3.2 Accurate and Efficient Techniques

Although above describes the step of setting up the DG method, it does not tell about the difficulties along the path. The Vandermonde matrix V and its inverse plays an important role in the creation of the nodal interpolating basis functions, L_j. The Vandermonde matrix depends on the basis p_i and on the nodal set. The nodal set is chosen to ensure well-behaved interpolating polynomials and avoid problems illustrated in Figure 3.2. We have not put any restrictions on the basis p_i, apart from it must span the polynomial space P^n. So far we have chosen the polynomials from the Pascal triangle of order n, but it is well known that the resulting Vandermonde matrix has a condition number growing exponentially with n. The also well known solution is to choose a orthonormalized basis to assure complete linear independence between the basis functions. A well-conditioned orthonormal basis can be found in [18, 43, 57] and has the form

$$\psi_i = \frac{\tilde{\psi}_i}{\sqrt{\gamma_i}}, \tag{3.63}$$

where

$$\tilde{\psi}_i = P^{(0,0)}_{\alpha_1}(\tilde{r})\left(\frac{1-\tilde{s}}{2}\right)^{\alpha_1} P^{(2\alpha_1+1,0)}_{\alpha_2}(\tilde{s}), \tag{3.64a}$$

$$\gamma_i = \left(\frac{2}{2\alpha_1+1}\right)\left(\frac{2^{2\alpha_1+2}}{2(\alpha_1+\alpha_2)+2}\right), \tag{3.64b}$$

$$\tilde{r} = \frac{2(1+r)}{1-s}, \quad \tilde{s} = s \ , \tag{3.64c}$$

$$i = i(\alpha_1, \alpha_2), \qquad \alpha_1, \alpha_2 \geq 0, \qquad \alpha_1 + \alpha_2 \leq n \tag{3.64d}$$

and $P_n^{(\alpha,\beta)}(x)$ is the classical Jacobi polynomial of order n. We will define a new Vandermonde matrix based on these basis functions, $(\mathsf{V})_{ij} = \psi_j(\mathbf{x}_i)$, and we will have exactly the relation Equation (3.51) for the nodal interpolating basis functions L_j.

Consider the operator $\hat{\mathsf{M}}$, inserting the definition of L_i in (3.57a) to get

$$\begin{aligned}(\hat{\mathsf{M}})_{ij} &= \int_{\mathsf{D}} \sum_{k=0}^{N} a_{kj}\psi_k \sum_{l=0}^{N} a_{li}\psi_l \ \mathrm{d}\mathbf{x} \\ &= \sum_{k=0}^{N}\sum_{l=0}^{N} a_{kj}a_{li} \int_{\mathsf{D}} \psi_k\psi_l \ \mathrm{d}\mathbf{x} \\ &= \sum_{k=0}^{N}\sum_{l=0}^{N} a_{kj}a_{li}\delta_{kl} \\ &= \sum_{k=0}^{N} a_{kj}a_{ki}, \end{aligned} \tag{3.65}$$

utilizing that the basis functions ψ_i are orthonormal. In matrix form above can be written as $\hat{\mathsf{M}} = \mathsf{A}^T\mathsf{A}$, and using the definition $\mathsf{A} = \mathsf{V}^{-1}$ we get

$$\hat{\mathsf{M}} = \left(\mathsf{V}\mathsf{V}^T\right)^{-1}. \tag{3.66}$$

Doing the same for Equation (3.57b)

$$\begin{aligned}(\hat{\mathsf{S}})_{ij} &= \int_{\mathsf{D}} \nabla \left(\sum_{k=0}^{N} a_{kj}\psi_k\right) \sum_{l=0}^{N} a_{li}\psi_l \ \mathrm{d}\mathbf{x} \\ &= \sum_{k=0}^{N}\sum_{l=0}^{N} a_{kj}a_{li} \int_{\mathsf{D}} \nabla\psi_k\psi_l \ \mathrm{d}\mathbf{x} \end{aligned} \tag{3.67}$$

and defining $s_{lk} = \int_{\mathsf{D}} \nabla\psi_k\psi_l \ \mathrm{d}\mathbf{x}$ we get

$$(\hat{\mathsf{S}})_{ij} = \sum_{k=0}^{N}\sum_{l=0}^{N} a_{li}s_{lk}a_{kj}. \tag{3.68}$$

Setting $(\mathsf{S}_\psi)_{lk} = s_{lk}$ we get in matrix form

$$\mathsf{S} = \mathsf{A}^T\mathsf{S}_\psi\,\mathsf{A} = (\mathsf{V}^{-1})^T\mathsf{S}_\psi\mathsf{V}^{-1} \tag{3.69}$$

Finally Equation (3.57c): Nodes on the edges are Legendre-Gauss-Lobatto quadrature nodes, hence the boundary integral operator $\hat{\mathsf{F}}$ can be expressed as

$$\hat{\mathsf{F}} = \mathsf{R}^T\ \hat{\mathsf{F}}^e\ \mathsf{R} \tag{3.70}$$

where R extracts the nodes on the element edges, and $\hat{\mathsf{F}}^e$ performs the 1D integration over the edges,

$$\hat{F}^e_{ij} = \int_{edge} l_j \; l_i \; \mathrm{d}\mathbf{x}, \tag{3.71}$$

l_i being the 1D Lagrangian interpolation polynomial defined by the nodes on the edge. Equation (3.71) is alike Equation (3.57a), though only in one dimension, hence it can be computed by a one dimensional analogue to Equation (3.66).

Notice how this formulation of $\hat{\mathsf{F}}$ simplifies the evaluation of the boundary integral: We only need values on the nodes of the edge. Hence to calculate the boundary integral of the numerical flux, which depends on the local and the neighboring element, we only need to pick out the matching nodes on the edge, e.g.

$$\hat{\mathsf{F}}(\mathbf{q}^- + \mathbf{q}^+) = \mathsf{R}^T \; \hat{\mathsf{F}}^e \; (\mathsf{R}^+\mathbf{q}^+ + \mathsf{R}^-\mathbf{q}^-) \tag{3.72}$$

where R^- and R^+ pick out matching nodes on the edge from the local element $\mathbf{q}^-$ and neighboring element $\mathbf{q}^+$ respectively.

Example 3.3: Matching nodes
Consider a domain with 2 elements as in Figure 3.9. Let element 1 be the local node and

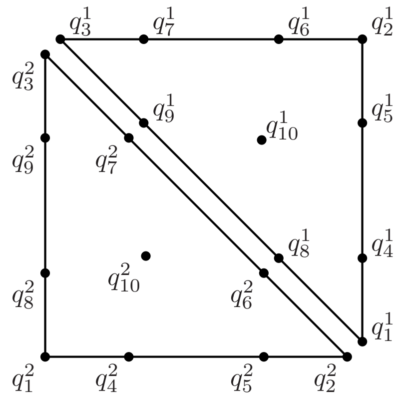


Figure 3.9: Numerical flux - matching nodes

element 2 the neighboring node, then

$$\mathbf{q}^+ = (q^1_0, ..., q^1_{10})^T \tag{3.73a}$$

$$\mathbf{q}^- = (q^2_0, ..., q^2_{10})^T \tag{3.73b}$$

and

$$\mathsf{R}^+\mathbf{q}^+ = \begin{pmatrix} q^1_1 \\ q^1_3 \\ q^1_8 \\ q^1_9 \end{pmatrix}, \qquad \mathsf{R}^-\mathbf{q}^- = \begin{pmatrix} q^2_2 \\ q^2_3 \\ q^2_6 \\ q^2_7 \end{pmatrix} \tag{3.74}$$

where now each position in the two vectors match the same position in space, and it can be put directly into Equation 3.72.

End of Example 3.3.

Notice that we may calculate S_ψ once and for all, and for a given nodal distribution we can determine the operators needed using the Vandermonde matrix and Equations (3.66), (3.69), and (3.70).

A similar description can be found in [37].

3.3.3 Using the Standard Element

We have now shown how to set up the method and create the operators on the standard element D. Returning to the global mesh with elements D^k, we shall now derive the operators on D^k, utilizing the smooth bijective mapping (3.43). On each D^k we want to approximate the conservation law weakly

$$\int_{\mathsf{D}^k} \left(\frac{\partial \tilde{q}}{\partial t} + \nabla \cdot \tilde{\mathbf{f}} \right) L_i(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 0, \quad \forall\ i. \tag{3.75}$$

which is the D^k equivalent to Equation (3.53). Following the procedure from Section 3.3.1, performing the same steps, we end up with equations for the operators

$$(\mathsf{M}^k)_{ij} = \int_{\mathsf{D}^k} L_j L_i \, \mathrm{d}\mathbf{x}, \tag{3.76a}$$

$$(\mathsf{S}^k)_{ij} = \int_{\mathsf{D}^k} \nabla L_j L_i \, \mathrm{d}\mathbf{x}, \tag{3.76b}$$

$$(\mathsf{F}^k)_{ij} = \oint_{\partial \mathsf{D}^k} L_j L_i \, \mathrm{d}\mathbf{x}, \tag{3.76c}$$

where the ˆ in Equations (3.57) are to indicate operators on the standard element D. Applying the bijective mapping, the integration domain becomes D, and we need to add a volume/line scaling, namely the absolute value of the determinant of the transformation Jacobian,

$$(\mathsf{M}^k)_{ij} = \int_{\mathsf{D}} L_j L_i \, |\mathsf{J}^k| \, \mathrm{d}\mathbf{x}, \tag{3.77a}$$

$$(\mathsf{S}^k)_{ij} = \int_{\mathsf{D}} \mathsf{J}^k \nabla_r L_j L_i \, |\mathsf{J}^k| \, \mathrm{d}\mathbf{x}, \tag{3.77b}$$

$$(\mathsf{F}^k)_{ij} = \oint_{\partial \mathsf{D}} L_j L_i \, |\mathsf{J}^k_e| \, \mathrm{d}\mathbf{x}, \tag{3.77c}$$

where now J^k is the transformation Jacobian from Equation (3.43) for the k'th element, J^k_e is similar for the line integral, and

$$\nabla = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} = \begin{pmatrix} r_x & r_y \\ s_x & s_y \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \end{pmatrix} = \mathsf{J}^k \, \nabla_r \tag{3.78}$$

is the chain rule for differentiation. Since now the transformation Jacobian is constant, then this is nothing more than

$$\mathsf{M}^k = |\mathsf{J}^k|\ \hat{\mathsf{M}}\ , \tag{3.79a}$$

$$\mathsf{S}^k = |\mathsf{J}^k|\ \mathsf{J}^k\ \hat{\mathsf{S}}\ , \tag{3.79b}$$

$$\mathsf{F}^k = |\mathsf{J}^k_e|\ \hat{\mathsf{F}}\ . \tag{3.79c}$$

Note: This is where using straight sided triangular elements make it considerably more simple compared to curvilinear triangular elements. Consider for example the operator M^k, when the Jacobian depends on space

$$\begin{aligned}
(\mathsf{M}^k)_{ij} &= \int_{\mathsf{D}} L_j L_i\ |\mathsf{J}^k|\ \mathrm{d}\mathbf{x}, \\
&= \int_{\mathsf{D}} \sum_{m=0}^{N} a_{mj}\psi_m \sum_{l=0}^{N} a_{li}\psi_l\ |\mathsf{J}^k|\ \mathrm{d}\mathbf{x} \\
&= \sum_{m=0}^{N}\sum_{l=0}^{N} a_{mj}a_{li} \int_{\mathsf{D}} \psi_m\psi_l|\mathsf{J}^k|\ \mathrm{d}\mathbf{x} \\
&= \sum_{k=0}^{N}\sum_{l=0}^{N} a_{kj}a_{li}w_{kl}
\end{aligned} \tag{3.80}$$

or in matrix form

$$\mathsf{M}^k = \mathsf{A}^T\mathsf{W}\mathsf{A} = (\mathsf{V}^{-1})^T\mathsf{W}(V)^{-1} \tag{3.81}$$

hence we need to compute the weights

$$w_{lm} = \int_{\mathsf{D}} \psi_m\psi_l|\mathsf{J}^k|\ \mathrm{d}\mathbf{x} \tag{3.82}$$

for each element D^k.

Using Equations (3.76) in the D^k equivalent of Equation (3.56) we get

$$\mathsf{M}^k\frac{\partial\hat{\mathbf{q}}^k}{\partial t} + \mathsf{S}^k\cdot\hat{\mathbf{f}}^k = \mathsf{F}(\mathbf{n}\cdot(\hat{\mathbf{f}}^k - \hat{\mathbf{f}}^*_k)) \tag{3.83}$$

or

$$|\mathsf{J}^k|\ \hat{\mathsf{M}}\frac{\partial\hat{\mathbf{q}}^k}{\partial t} + |\mathsf{J}^k|\ (\mathsf{J}^k\ \hat{\mathsf{S}})\cdot\hat{\mathbf{f}}^k = |\mathsf{J}^k_e|\ \hat{\mathsf{F}}(\mathbf{n}\cdot(\hat{\mathbf{f}}^k - \hat{\mathbf{f}}^*_k)). \tag{3.84}$$

We now only need to specify the numerical flux $\hat{\mathbf{f}}^*_k$ in order to solve the problem. Using a central flux, i.e.

$$\hat{\mathbf{f}}^*_k(\hat{\mathbf{q}}^-,\hat{\mathbf{q}}^+) = \frac{1}{2}(\mathbf{f}(\hat{\mathbf{q}}^-) + \mathbf{f}(\hat{\mathbf{q}}^+)) = \frac{1}{2}(\hat{\mathbf{f}}^- + \hat{\mathbf{f}}^+)\ , \tag{3.85}$$

and since $\hat{\mathbf{f}}^k = \hat{\mathbf{f}}^-$ we get

$$|\mathsf{J}^k|\ \hat{\mathsf{M}}\frac{\partial\hat{\mathbf{q}}^k}{\partial t} + |\mathsf{J}^k|\ (\mathsf{J}^k\ \hat{\mathsf{S}})\cdot\hat{\mathbf{f}}^k = |\mathsf{J}^k_e|\ \hat{\mathsf{F}}(\mathbf{n}\cdot(\frac{1}{2}(\hat{\mathbf{f}}^- - \hat{\mathbf{f}}^+))) \tag{3.86}$$

3.3.4 Putting It All Together - Efficiently

Collecting node values of each element $\hat{\mathbf{q}}_k$ in a vector $\hat{\mathbf{q}}$,

$$\hat{\mathbf{q}} = (\hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2, ..., \hat{\mathbf{q}}_K)^T \tag{3.87}$$

and similar for $\hat{\mathbf{f}}$. Define R_k^- to pick out node values on the edge of the element k from the matrix $\hat{\mathbf{f}}$, and R_k^+ picks out matching nodes from neighboring elements of element k from $\hat{\mathbf{f}}$, and $(\mathsf{R}^-)^T$ to put back the node values on the edge into the standard element vector $\hat{\mathbf{q}}_k$, then we get

$$|\mathsf{J}^k|\ \hat{\mathsf{M}}\frac{\partial \hat{\mathbf{q}}^k}{\partial t} + |\mathsf{J}^k|\ (\mathsf{J}^k\ \hat{\mathsf{S}}) \cdot \hat{\mathbf{f}}^k = |\mathsf{J}_e^k|\ (\mathsf{R}^-)^T \hat{\mathsf{F}}^e(\mathbf{n} \cdot (\frac{1}{2}(\mathsf{R}_k^- \hat{\mathbf{f}}^k - \mathsf{R}_k^+ \hat{\mathbf{f}}^k))) \tag{3.88}$$

To solve above problem, we need to precompute the operators on the standard element, hence for a given nodal set:

- $\hat{\mathsf{M}}$, mass matrix
- $\hat{\mathsf{S}}$, stiffness matrix
- $\hat{\mathsf{F}}^e$, element edge integration matrix
- $(\mathsf{R}^-)^T$, from edge nodes to element nodes

For a given triangular decomposition, we need to precompute for each element D_k

- J^k, the smooth bijective mapping
- $|\mathsf{J}^k|$ and $|\mathsf{J}_e^k|$ scaling factors
- $\mathbf{n}$, the outward pointing normal vector
- R_k^- and R_k^+.

The creation of $\hat{\mathsf{F}}^e$, $(\mathsf{R}^-)^T$, R_k^-, and R_k^+ must match in order to compute the edge integral correctly for all elements, which require some rather difficult logics.

If we furthermore define and precompute

$$\mathsf{S} = \hat{\mathsf{M}}^{-1}\ \hat{\mathsf{S}} \tag{3.89}$$

$$\mathsf{F}^e = \hat{\mathsf{M}}^{-1} (\mathsf{R}^-)^T\ \hat{\mathsf{F}}^e \tag{3.90}$$

$$\mathsf{F}_k = \frac{1}{2}\ \frac{|\mathsf{J}_e^k|}{|\mathsf{J}^k|} \tag{3.91}$$

we end up with

$$\frac{\partial \hat{\mathbf{q}}^k}{\partial t} + (\mathsf{J}^k\ \mathsf{S}) \cdot \hat{\mathbf{f}}^k = \mathsf{F}_k \mathsf{F}^e(\mathbf{n} \cdot (\mathsf{R}_k^- \hat{\mathbf{f}}^k - \mathsf{R}_k^+ \hat{\mathbf{f}}^k)). \tag{3.92}$$

This is the most compact form of the scheme, and in most cases as well the most efficient as the most convenient form.

Similar forms exists for other numerical fluxes than the central flux. The general form is

$$\frac{\partial \hat{\mathbf{q}}^k}{\partial t} + (\mathsf{J}^k\ \mathsf{S}) \cdot \hat{\mathbf{f}}^k = \mathsf{F}_k \mathsf{F}^e(\mathbf{n} \cdot (2(\mathsf{R}_k^- \hat{\mathbf{f}}^k - \hat{\mathbf{f}}_k^*))). \tag{3.93}$$

where the number 2 balances the $\frac{1}{2}$ in F_k. Inserting a Lax-Friedrich flux, we would then get

$$\frac{\partial \hat{\mathbf{q}}^k}{\partial t} + (\mathsf{J}^k\,\mathsf{S}) \cdot \hat{\mathbf{f}}^k = \mathsf{F}_k \mathsf{F}^e(\mathbf{n} \cdot (\mathsf{R}_k^- \hat{\mathbf{f}}^k - \mathsf{R}_k^+ \hat{\mathbf{f}}^k - c(\mathsf{R}_k^+ \hat{\mathbf{q}}^k - \mathsf{R}_k^- \hat{\mathbf{q}}^k)))\,. \tag{3.94}$$

Note that the sloppy notation is still in use, i.e.

$$(\mathsf{J}^k\,\mathsf{S}) \cdot \hat{\mathbf{f}}^k = \left(\begin{bmatrix} r_x & r_y \\ s_x & s_y \end{bmatrix} \begin{bmatrix} \mathsf{S}_r \\ \mathsf{S}_s \end{bmatrix} \right) \cdot \begin{bmatrix} \mathbf{f}_x^k \\ \mathbf{f}_y^k \end{bmatrix}, \tag{3.95a}$$

$$\mathbf{n} \cdot (\mathsf{R}_k^- \hat{\mathbf{f}} - \mathsf{R}_k^+ \hat{\mathbf{f}}) = \begin{bmatrix} \mathbf{n}_x \\ \mathbf{n}_y \end{bmatrix} \cdot \begin{bmatrix} \mathsf{R}_k^- \hat{\mathbf{f}}_x - \mathsf{R}_k^+ \hat{\mathbf{f}}_x \\ \mathsf{R}_k^- \hat{\mathbf{f}}_y - \mathsf{R}_k^+ \hat{\mathbf{f}}_y \end{bmatrix}. \tag{3.95b}$$

3.3.5 Linear Wave – Implementation Example

Having presented the details of the DG method in 2D, we shall show an implementation in Matlab utilizing the DG method. The example is included to show that even though the derivation of the method is fairly complex, the implementation is surprisingly simple. We shall solve the linear wave equation,

$$\frac{\partial^2}{\partial t^2} \eta = \nabla \cdot \nabla \eta\,. \tag{3.96}$$

We rewrite it as a first order conservation law,

$$\frac{\partial}{\partial t} \mathbf{u} = \nabla \eta \quad \Leftrightarrow \quad \begin{cases} \frac{\partial}{\partial t} u = \frac{\partial}{\partial x} \eta \\ \frac{\partial}{\partial t} v = \frac{\partial}{\partial y} \eta \end{cases}, \tag{3.97a}$$

$$\frac{\partial}{\partial t} \eta = \nabla \cdot \mathbf{u} = \frac{\partial}{\partial x} u + \frac{\partial}{\partial y} v\,. \tag{3.97b}$$

Table 3.1 lists the Matlab implementation using the DG method on the form of Equation (3.92), and a fourth order low storage Runge Kutta method [10] to integrate in time. The part concerning the DG method is emphasized, by typesetting the Runge Kutta part in gray.

We shall not go over every detail of the implementation, which also does not show how to set up the grid and initial conditions. But we will describe the vital parts concerning the DG-implementation, to show how efficiently it may be implemented.

In the implementation, the solution vector for element k, u^k, is collected in a matrix `U`, such that the kth column of `U` contain the u^k values. Hence the computation of $\frac{\partial}{\partial r} u$, which in element k is accomplished by the matrix-vector product $\mathsf{S}_r \mathbf{u}^k$, can for all elements be computed by the matrix-matrix product S_r`U`.

Lets us go through the details in the calculation of $\nabla \eta$, i.e., `dEdx` and `dEdy`, used in the equation for $\mathbf{u}$, i.e.,

$$\frac{\partial \hat{\mathbf{u}}^k}{\partial t} = (\mathsf{J}^k\,\mathsf{S}) \hat{\eta}^k + \mathsf{F}_k \mathsf{F}^e(\mathbf{n}(\mathsf{R}_k^+ \hat{\eta}^k - \mathsf{R}_k^- \hat{\eta}^k))\,. \tag{3.98}$$

```
for tstep = 1:Nsteps
  for INTRK = 1:5   % Runge-Kutta stage loop

    % initiate Runge-Kutta stage
    RKtime = time+dt*rk4c(INTRK);
    resU   = rk4a(INTRK)*resU;
    resV   = rk4a(INTRK)*resV;
    resE   = rk4a(INTRK)*resE;

    % Calculate derivatives - the DG part
    ff   = umNtoF*U; flux(:) = ff(mapR) -ff(:);
    dUdx = rx.*(umDr*U)+sx.*(umDs*U) + umFtoN*(Fscale.*(nx.*flux));

    ff   = umNtoF*V; flux(:) = ff(mapR) -ff(:);
    dVdy = ry.*(umDr*V)+sy.*(umDs*V) + umFtoN*(Fscale.*(ny.*flux));

    ff   = umNtoF*E; flux(:) = ff(mapR) -ff(:);
    dEdx = rx.*(umDr*E)+sx.*(umDs*E) + umFtoN*(Fscale.*(nx.*flux));
    dEdy = ry.*(umDr*E)+sy.*(umDs*E) + umFtoN*(Fscale.*(ny.*flux));

    % update RK-residual
    resU = resU - dt*(dEdx        );
    resV = resV - dt*(dEdy        );
    resE = resE - dt*((dUdx+dVdy) );

    % finish Runge-Kutta stage
    U = U + rk4b(INTRK)*resU;
    V = V + rk4b(INTRK)*resV;
    E = E + rk4b(INTRK)*resE;

  end;
  time = time+dt;   % update time
end
```

Table 3.1: Matlab implementation - DG part emphasized

Consider the flux,

$$flux = \mathsf{R}^{+}\hat{\eta} - \mathsf{R}^{-}\hat{\eta} \quad \Leftrightarrow \quad \texttt{ff=umNtoF*E; flux(:)=ff(mapR)-ff(:);} \tag{3.99}$$

where `ff=umNtoF*E` collects all nodes on the edges in `ff`, and `mapR` picks out the matching nodes on the edge of the neighboring element. The integration over the boundary of the flux is performed as

$$\mathsf{F}_k\mathsf{F}^e(\mathbf{n}\cdot(flux)) \quad \Leftrightarrow \quad \begin{cases} \texttt{umFtoN*(Fscale.*(nx.*flux))} \\ \texttt{umFtoN*(Fscale.*(ny.*flux))} \end{cases}, \tag{3.100}$$

where F_k = `Fscale` and F^e = `umFtoN`. Finally, we calculate the differentiation,

$$(\mathsf{J}^k\ \mathsf{S})\hat{\eta}^k \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \texttt{rx.*(umDr*E)+sx.*(umDs*E)} \\ \texttt{ry.*(umDr*E)+sy.*(umDs*E)} \end{array} \right. , \tag{3.101}$$

where S_r = `umDr` and S_s = `umDs`. The implementation uses the matrix-matrix operation for $\frac{\partial}{\partial r}\eta$ and $\frac{\partial}{\partial s}\eta$, and subsequently applies the scaling according to the chain rule, Equation (3.95a), for each element.

This implementation is matrix free, in the sense that the differentiation matrices are never created explicitly. Every operation is based on the standard triangular element, and applied on each element individually using the transformation Jacobian J^k. If we wanted to simulate the same equations on another domain, then we would need to recompute the map `mapR` and the scaling factors `Fscale`, `nx`, `ny`, `rx`, `ry`, `sx`, and `sy`, while the remaining part of the implementation in Table 3.1 would be exactly the same. Notice that all the scaling factors can be computed locally and independently for each element.

We shall also give an example of the accuracy of the method. We shall solve the wave equation in a periodic domain of size $[-0.5; 0.5] \times [-0.5; 0.5]$ with initial conditions

$$\mathbf{u}(x, y, 0) = \mathbf{0} \tag{3.102}$$

$$\eta(x, y, 0) = \cos(2\pi x)\cos(2\pi y) \tag{3.103}$$

having the exact solution

$$\eta(x, y, t) = \cos(\sqrt{8}\pi t)\cos(2\pi x)\cos(2\pi y)\,. \tag{3.104}$$

This we have solved in a domain having 62 elements, the mesh can be seen in Figure D.1 in Appendix. Initial conditions are plotted in Figure 3.10. The error after one period is presented in Table 3.2, for different orders of the polynomial basis P^n. The table also show the number of unknowns in the system, #unknowns, and the number of time steps, `Nsteps`, needed to compute one full period. The table

P^n	#unknowns	`Nsteps`	$\max\lvert\eta - \eta_{exact}\rvert$
3	620	25	8.9-03
5	1302	50	1.1-04
7	2232	100	6.4-07
9	3410	150	3.3-09

Table 3.2: Error of linear wave equation after one period

verifies our statement, that the DG method is very accurate for smooth problems.

3.3.6 Filtering of Nonlinear Problems

For problems which are no longer linear, the nonlinear interaction of modes results in aliasing and leads to unbounded growth of the modes in time [35]. Aliasing is

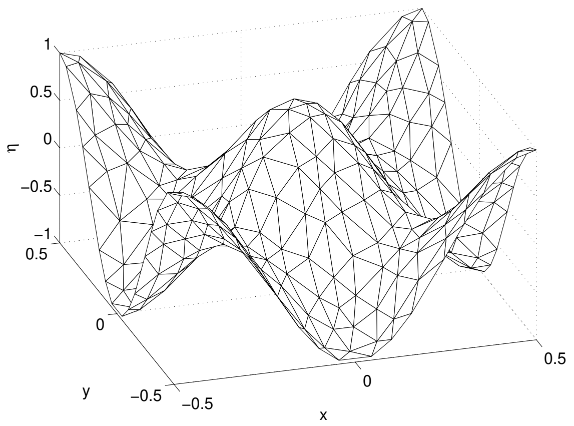


Figure 3.10: Solution to linear wave equation

the phenomenon, where high modes in an under-resolved solution are represented by the same nodal values as lower a mode, adding up on the original lower mode.

The problem can be illustrated considering two interpolating polynomial approximations each of degree N

$$\tilde{u}(x,t) = \sum_{i=0}^{N} \hat{u}_i(t)\phi_i(x) \; , \tag{3.105a}$$

$$\tilde{v}(x,t) = \sum_{i=0}^{N} \hat{v}_i(t)\phi_i(x) \; . \tag{3.105b}$$

The product of the two gives

$$\begin{aligned}\tilde{u}(x,t)\,\tilde{v}(x,t) &= \sum_{i=0}^{N}\sum_{j=0}^{N} \hat{u}_i(t)\hat{v}_j(t)\phi_i(x)\phi_j(x) \\ &= \sum_{i=0}^{2N} \hat{(uv)}_i(t)\phi_i(x) \; ,\end{aligned} \tag{3.106}$$

implying that we now have a polynomial of degree $2N$.

Having a nodal setup $\phi_i(x) = L_i(x)$, it is very convenient to multiply the nodal values, i.e. defining the product as

$$\tilde{u}(x,t)\,\tilde{v}(x,t) = \sum_{i=0}^{N} \hat{v}_i(t)\hat{u}_i(t)L_i(x) \; . \tag{3.107}$$

Equations (3.106) and (3.107) will give the same value on the nodes, hence in the nodal setup it makes sense. However, outside the nodes they will not equal.

Equation (3.107) can be considered as a projection of the result of Equation (3.106) to the space P_N spanned by $\{L_j\}$. However, this projection will transfer energy of the high modes to lower modes, adding up on the true value of the lower mode, which then may lead to the unbounded growth of modes. The aliasing affects primarily the high modes of the space P^N.

We shall use a low-pass filter damping each mode of the solution, damping mostly on the high modes. Consider the exponential filter damping mode m of total M modes

$$\sigma\left(\frac{m}{M}\right) = \begin{cases} 1 & 0 \le m \le M_c \\ \exp\left(-\alpha\left(\frac{m-M_c}{M-M_c}\right)^p\right) & M_c < m \le M \end{cases} \tag{3.108}$$

where p is the order of the filter, $\alpha = -\log(\varepsilon)$, ε is machine precision, and M_c is a start cutoff mode. Hence modes $m \le M_c$ are left untouched and modes $m > M_c$ are damped according to the order p. Figure 3.11 shows the filter values for different

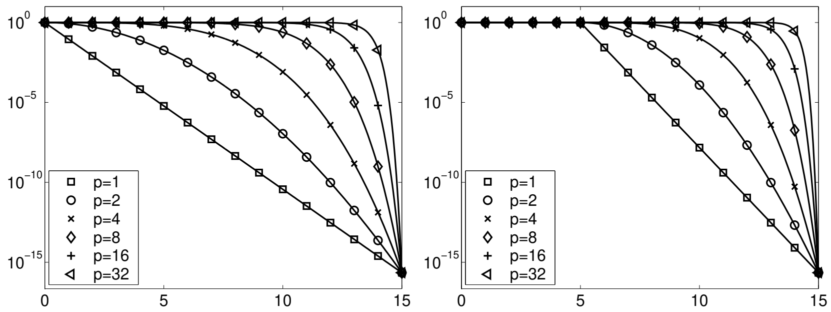


Figure 3.11: Exponential filter for $M = 15$, $M_c = 0$ (left) and $M_c = 5$ (right) for different filter orders p.

modes m and start cutoff modes M_c. Choosing $\alpha = -\log(\varepsilon)$ effectively implies that the highest modes are set to zero.

Consider an interpolating approximation

$$\tilde{u} = \sum_{n=0}^{N} \hat{u}_n \phi_n(x)\,. \tag{3.109}$$

Where the index n corresponds to increasing mode, i.e. 1D trigonometric polynomials $\phi_n(x) = \exp(inx)$, ordinary polynomials $\phi_n(x) = x^n$, or Chebyshev polynomials $\phi_n(x) = T_n(x)$, we shall use $m = n$ and $M = N$. Then the filtering works as follows,

$$\mathcal{F}\tilde{u} = \sum_{n=0}^{N} \sigma\left(\frac{n}{N}\right) \hat{u}_n \phi_n(x). \tag{3.110}$$

hence leaving modes $n \leq M_c$ and damping higher modes depending on the order of the filter. Note for the 2D orthonormal basis based on Jacobi polynomials $\psi_k(x)$ from Section 3.3.2, the mode is not directly given by the index k but by $m = \alpha_1 + \alpha_2$. Let in general $\psi_k(x)$ denote the k'th basis-function having mode $m = m(k)$ of total M modes.

We want to implement the filtering efficiently, i.e. using the nodal setup from previous sections. We can not use the filter directly on each nodal basis function L_j, since they each contain many different modes. We can however filter on the ψ_k's, hence we shall exploit the identity $L_i = \sum_{k=0}^{N} a_{ki}\psi_k$. To filter a nodal interpolating approximation

$$\begin{aligned} \mathcal{F}\tilde{u} &= \mathcal{F}\sum_{i=0}^{N} \hat{u}_i L_i = \mathcal{F}\sum_{i=0}^{N} \hat{u}_i \sum_{k=0}^{N} a_{ki}\psi_k^n = \mathcal{F}\sum_{k=0}^{N}\sum_{i=0}^{N} a_{ki}\hat{u}_i\psi_k \\ &= \sum_{k=0}^{N}\sum_{i=0}^{N} \sigma\left(\frac{m(k)}{M}\right) a_{ki}\hat{u}_i\psi_k \end{aligned} \tag{3.111}$$

We want the result on the nodal basis L_l, hence exploiting that $\psi_k = \sum_{l=0}^{N} b_{lk}L_l$, where $(\mathsf{V})_{lk} = b_{lk}$ are exactly the elements of the Vandermonde matrix, we get

$$\begin{aligned} \mathcal{F}\tilde{u} &= \sum_{k=0}^{N}\sum_{i=0}^{N} \sigma\left(\frac{m(k)}{M}\right) a_{ki}\hat{u}_i \sum_{l=0}^{N} b_{lk}L_l \\ &= \sum_{l=0}^{N}\sum_{k=0}^{N}\sum_{i=0}^{N} b_{lk}\,\sigma\left(\frac{m(k)}{M}\right) a_{ki}\hat{u}_i L_l \\ &= \sum_{l=0}^{N} \underline{\hat{u}}_l L_l \end{aligned} \tag{3.112}$$

where

$$\underline{\hat{u}}_l = \sum_{k=0}^{N}\sum_{i=0}^{N} b_{lk}\,\sigma\left(\frac{m(k)}{M}\right) a_{ki}\hat{u}_i \tag{3.113}$$

is the filtered nodal values. In matrix notation

$$\underline{\hat{\mathbf{u}}} = \mathsf{V}\,\Sigma\,\mathsf{A}\,\hat{\mathbf{u}} = \mathsf{H}\,\hat{\mathbf{u}}\,. \tag{3.114}$$

where Σ is a diagonal matrix with elements $(\Sigma)_{kk} = \sigma(m(k)/M)$. Since now $\mathsf{A} = \mathsf{V}^{-1}$, we get the filtering matrix operator

$$\mathsf{H} = \mathsf{V}\,\Sigma\,\mathsf{V}^{-1}\,. \tag{3.115}$$

Notice how this again is efficiently and elegantly implemented using the Vandermonde matrix.

3.4 Elliptic Problems

A subproblem when solving the incompressible Navier Stokes equations is a density weighted Poisson problem with Neumann boundary conditions,

$$\nabla \cdot \frac{1}{\rho} \nabla p = f \quad \text{in } \Omega \tag{3.116a}$$

$$n \cdot \nabla p = 0 \quad \text{on } \partial\Omega, \tag{3.116b}$$

where ρ is a function of space $\mathbf{x}$. This is not a problem for which the DG method was originally intended.

DG methods have been used to solve elliptic problems, [3, 6, 12, 14, 16]. Traditionally the operators have been derived using the weak form of the DG method. We will here derive the operators using a combination of the weak and the strong form, and include the density weighting in the operator.

A characteristic of this pure Neumann problem is that it is singular. It has a null-space of dimension one, and the null vector is the constant vector. This means that for a solution p_a, we can add an arbitrary constant, $p_b = p_a + c$, and p_b will also be a solution. The solution is only unique up till a constant. Furthermore, there is not a solution to any right hand side f, the right hand side f needs to be consistent to ensure solvability at all. When solving the incompressible Navier Stokes equations, the right hand side f is not in general consistent, giving the system no solution. To get a solution, we need to ensure that the right hand side is consistent.

Define $\mathcal{L} = \nabla \cdot \frac{1}{\rho} \nabla$, let $R(\mathcal{L})$ and $N(\mathcal{L})$ denote the range and null space of $\mathcal{L}$. The right hand side is consistent when $f \in R(\mathcal{L})$, i.e. when f can be "hit" by $\mathcal{L}\ p$ for some p. Remember that $R(\mathcal{L}) = N(\mathcal{L}^T)^{\perp}$. If $\mathcal{L}$ is symmetric, then $R(\mathcal{L}) = N(\mathcal{L})^{\perp}$. The null space of $\mathcal{L}$ is known to be the constant vector, $N(\mathcal{L}) = \mathbf{1}$. Hence, in the symmetric case we remove from f the part existing in $N(\mathcal{L})$, i.e. we use

$$f^c = f - \mathbf{1}\ (\mathbf{1} \cdot f)/(\mathbf{1} \cdot \mathbf{1}),$$

to ensure solvability. In the non-symmetric case, defining $\mathbf{m} = N(\mathcal{L}^T)$, we recover

$$f^c = f - \mathbf{m}\ (\mathbf{m} \cdot f)/(\mathbf{m} \cdot \mathbf{m}).$$

The procedure of determining $\mathbf{m}$ is in general quite costly. In a free surface flow, the surface moves in each step, hence ρ changes in each step, and one would need to recalculate $\mathbf{m}$ in each step.

A simple approach to solve the null space problem is to set a Dirichlet boundary condition at one node in space, thereby defining the arbitrary constant and removing the null space. This approach has a number of side effects. The main one is that the Neumann condition is not fulfilled at that node, and it may produce a sink or a source point, making the water flow out of or into the domain from this node. This side effect has been verified experimentally.

However, by careful creation of the discrete operators, we have observed that we get $\mathbf{m} = \mathbf{1}$. This has been the experience before, i.e., when discretizing the

Laplace operator, even though the operator is not symmetric, careful creation can actually retain this property. In [24] the same was the case for a finite volume discretization of the Laplace operator, giving also an unsymmetric operator and also $\mathbf{m} = 1$. It was, however, unexpected that this also is the case for a density weighted operator based on the LDG method.

The standard procedure in DG methods [3, 6, 12, 14, 16] when discretizing the Laplace operator is to split the system into two first order equations

$$\nabla \cdot \mathbf{q} = f, \tag{3.117a}$$
$$\nabla p = \rho\, \mathbf{q}. \tag{3.117b}$$

This type of splitting are sometimes called Local discontinuous Galerkin, LDG. If we approximate the first in the weak DG sense and the second in the strong sense, following the procedure in Section 3.3.1 though omitting the details, and for convenience using the notation $(\cdot,\cdot)_{\mathsf{D}}$ and $(\cdot,\cdot)_{\partial\mathsf{D}}$ for the integrals we obtain

$$-\left(\mathbf{q}, \nabla L_j\right)_{\mathsf{D}} = \left(f, L_j\right)_{\mathsf{D}} - \left(\mathbf{n}\cdot\mathbf{q}^*, L_j\right)_{\partial\mathsf{D}} \tag{3.118a}$$
$$\left(\nabla p, L_j\right)_{\mathsf{D}} = \left(\rho\,\mathbf{q}, L_j\right)_{\mathsf{D}} + \left(\mathbf{n}(p-p^*), L_j\right)_{\partial\mathsf{D}} \tag{3.118b}$$

where p^* is the Dirichlet conditions and $\mathbf{n}\cdot\mathbf{q}^*$ is the Neumann conditions. On the global boundary we have only Neumann condition and the $(p - p^*)$-part can be discarded there. In the case of homogeneous Neumann conditions, we simply set $\mathbf{n}\cdot\mathbf{q}^* = 0$.

In terms of discrete operators on the standard element, we get

$$-\hat{\mathsf{S}}^T \mathbf{q} = \hat{\mathsf{M}}\mathbf{f} - \hat{\mathsf{F}}(\mathbf{n}\cdot\mathbf{q}^*) \tag{3.119a}$$
$$\hat{\mathsf{S}}\mathbf{p} = \hat{\mathsf{M}}(\rho\,\mathbf{q}) + \hat{\mathsf{F}}(\mathbf{n}(\mathbf{p}-\mathbf{p}^*)) \tag{3.119b}$$

Consider the simplified special case where ρ is constant and 1, and consider only one element, hence having Neumann conditions on all boundaries so the $\hat{\mathsf{F}}(\mathbf{n}(\mathbf{p}-\mathbf{p}^*))$ term drops out , then combining Equation (3.119a) and (3.119b) yields

$$-\hat{\mathsf{S}}^T\ \hat{\mathsf{M}}^{-1}\ \hat{\mathsf{S}}\ \mathbf{p} = \hat{\mathsf{M}}\mathbf{f} - \hat{\mathsf{F}}(\mathbf{n}\cdot\mathbf{q}^*). \tag{3.120}$$

Since $\hat{\mathsf{M}}$ is symmetric, then so is its inverse, hence $\hat{\mathsf{S}}^T\ \hat{\mathsf{M}}^{-1}\ \hat{\mathsf{S}}$ forms a symmetric system. In a multi-element setup, we require weak continuity between the elements, hence keeping the $\hat{\mathsf{F}}(\mathbf{n}(\mathbf{p}-\mathbf{p}^*))$, but the resulting system will still be symmetric as long as the numerical fluxes $\mathbf{p}^*$ and $\mathbf{q}^*$ are symmetric over element boundaries, i.e. the same flux function is used in neighboring elements. In this work we use central fluxes. Internal penalty fluxes [3] are equally efficient, although often leading to a slightly worse conditioning.

In the general case of ρ depending on space, the equation for one element corresponding to Equation 3.120 becomes

$$-\hat{\mathsf{S}}^T\ \frac{1}{\rho}\,\hat{\mathsf{M}}^{-1}\ \hat{\mathsf{S}}\ \mathbf{p} = \hat{\mathsf{M}}\mathbf{f} - \hat{\mathsf{F}}(\mathbf{n}\cdot\mathbf{q}^*), \tag{3.121}$$

which is not symmetric.

Note 1: If the discrete operators fulfill a discrete integration by parts rule,

$$(\nabla \cdot \mathbf{q}, L_j)_{\mathsf{D}} = -(\mathbf{q}, \nabla L_j)_{\mathsf{D}} + (\mathbf{n} \cdot \mathbf{q}, L_j)_{\partial\mathsf{D}},$$
$$\rightarrow \qquad \hat{\mathsf{S}}\,\mathbf{q} = -\hat{\mathsf{S}}^T\mathbf{q} + \hat{\mathsf{F}}\,(\mathbf{n} \cdot \mathbf{q}),$$

then Equation (3.117a) can be approximated also in a strong sense and produce exactly the same system as when approximated using weak sense. The strong version

$$(\nabla \cdot \mathbf{q}, L_j)_{\mathsf{D}} - (\mathbf{n} \cdot (\mathbf{q} - \mathbf{q}^*), L_j)_{\partial\mathsf{D}} = (f, L_j)_{\mathsf{D}}$$

becomes $\hat{\mathsf{S}}\,\mathbf{q} - \hat{\mathsf{F}}\,(\mathbf{n} \cdot \mathbf{q}) = \hat{\mathsf{M}}\,\mathbf{f} - \hat{\mathsf{F}}\,(\mathbf{n} \cdot \mathbf{q}^*)$. This is ensured by evaluating the operators as in [37].

Note 2: If solving this Poisson equation with Dirichlet or periodic boundary conditions, a stabilizing term controlling the null space is needed. In this case we use a standard penalty penalization technique [3].

3.4.1 Eigenvalues of the Elliptic Operator

We shall here consider the eigenvalues of the simplified elliptic operator where $\rho = 1$, i.e.

$$\nabla^2 u(x) = \lambda u(x), \qquad x \in [0;1] \times [0;1] \tag{3.122a}$$
$$u(x)|_{\partial\Omega} = 0 \tag{3.122b}$$

It is well known that the eigenvalues of the Laplace operator are $\lambda_{\alpha\beta} = -(\alpha^2 + \beta^2)\pi^2$, $\alpha, \beta = 1, ...$ A discrete version of the Laplace operator will approximate the lowest eigenvalues well. The largest absolute eigenvalue is however important when solving the system, since it tells something about how easy it is to solve.

It is well known that a second order finite difference or finite element approximation of the Laplace operator will have the largest absolute eigenvalue of size $\mathcal{O}(1/\Delta x^2)$, where Δx is the distance between grid-points. Spectral method are worse in the sense they have a larger absolute maximum eigenvalue

$$\lambda_{\max} = \mathcal{O}(P^4) \tag{3.123}$$

where $P = \frac{1}{\Delta x}$ is the polynomial degree used. See e.g. [17, 40] for details. For this spectral element method, we have a combination of the two, namely

$$\lambda_{\max} = \mathcal{O}(P^4/\Delta x^2) \tag{3.124}$$

where Δx is the element side length and P is the polynomial degree in the element.

Table 3.3 and Figure 3.12 shows the largest absolute eigenvalue of the spectral element method when discretized as in the previous section, or to be exact:

$$\hat{\mathsf{M}}^{-1}\hat{\mathsf{S}}^T\,\hat{\mathsf{M}}^{-1}\,\hat{\mathsf{S}}\,\mathbf{u} = \lambda\mathbf{u}\,. \tag{3.125}$$

P^n	0.2	0.1	0.05	0.025
2	1.5e+03	7.7e+03	2.2e+04	1.1e+05
3	4.7e+03	2.5e+04	7.2e+04	3.7e+05
4	1.1e+04	6.0e+04	1.8e+05	9.1e+05
5	2.3e+04	1.2e+05	3.7e+05	1.9e+06
6	4.4e+04	2.3e+05	7.0e+05	3.5e+06
7	7.5e+04	3.9e+05	1.2e+06	
8	1.2e+05	6.3e+05	1.9e+06	
9	1.9e+05	9.7e+05		
10	2.8e+05	1.4e+06		
11	4.0e+05			
12	5.5e+05			

Table 3.3: Maximum absolute eigenvalue of 2D Laplace operator for different polynomial orders

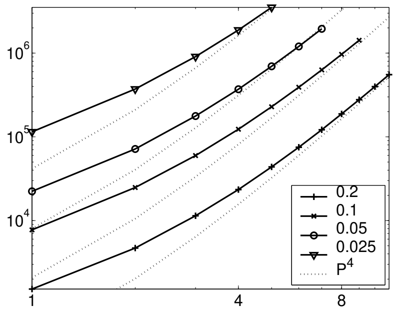


Figure 3.12: Maximum absolute eigenvalue of 2D Laplace operator for different polynomial orders P^n

The test was conducted on 4 different grid resolution having approximate side-lengths of order as specified in the table. In number of elements, in corresponds to 62, 226, 894, and 3602 elements for side-length $\Delta x = 0.2, 0.1, 0.05, 0.0025$ respectively. The tests left blank are due to memory limits. The figure verifies nicely the P^4 dependence of the polynomial order.

Table 3.4 and Figure 3.13 shows the error of the smallest eigenvalue $\lambda_1 = 2\pi^2$ for same setup as above, and verifies the fast convergence of the spectral method until the error reaches about 10^{-10} where rounding errors set in. Notice that the best results are achieved by increasing the polynomial order rather than decreasing the element size, which is also the expected behavior: For the smallest eigenvalue, the eigenvector is very smooth and represented extremely well using high order polynomials.

Similar results have been found for the Laplace operator with Neumann boundary conditions.

P^n	0.2	0.1	0.05	0.025
2	3.1e-01	9.7e-02	2.4e-02	9.7e-03
3	3.6e-02	4.9e-03	6.3e-04	7.6e-04
4	3.7e-04	3.9e-05	3.0e-06	5.2e-05
5	7.2e-05	2.7e-06	7.8e-07	6.6e-02
6	1.4e-06	2.6e-08	2.2e-07	
7	2.3e-07	6.7e-08	1.3e-07	
8	1.1e-07	2.0e-09	9.7e-08	
9	9.9e-11	3.8e-08		
10	4.7e-11	2.3e-07		
11	2.3e-10			
12	1.4e-09			

Table 3.4: Error of smallest eigenvalue $\lambda_0 = 2\pi^2$ for different polynomial orders

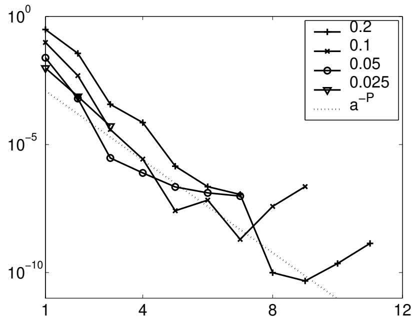


Figure 3.13: Error of smallest eigenvalue for different polynomial orders P^n

3.5 The Incompressible Navier Stokes Equations

We shall now derive the incompressible Navier Stokes in discrete form when applying the DG method. We will introduce the notation

$$(f, g)_{\mathsf{D}^k} = \int_{\mathsf{D}^k} f \; g \; \mathrm{d}\mathbf{x} \, , \qquad (f, g)_{\partial\mathsf{D}^k} = \int_{\partial\mathsf{D}^k \setminus \partial\Omega} f \; g \; \mathrm{d}\mathbf{x} \, ,$$

$$(f, g)_{\partial\Omega} = \int_{\partial\mathsf{D}^k \cap \partial\Omega} f \; g \; \mathrm{d}\mathbf{x} \, ,$$

thereby separating inner element boundaries from the global domain boundaries. Rewriting Equation (2.12), introducing the variable g, and leaving out the surface tension for now, we obtain

$$\frac{\partial u_i}{\partial t} + \nabla \cdot (\mathbf{u} u_i) = -\frac{1}{\rho} \nabla_i p + \frac{1}{Re} \frac{\mu}{\rho} \nabla \cdot \mathbf{g}_i \; , \tag{3.126a}$$

$$\nabla u_i = \mathbf{g}_i \; . \tag{3.126b}$$

Define the following:

$$N^k(u_i) = (\nabla \cdot (\mathbf{u}u_i), L_j)_{\mathsf{D}^k} - (\mathbf{n} \cdot (\mathbf{u}u_i - (\mathbf{u}u_i)^*), L_j)_{\partial \mathsf{D}^k} \ , \tag{3.127a}$$

$$L^k(\mathbf{g}) = (\nabla \cdot \mathbf{g}, L_j)_{\mathsf{D}^k} - (\mathbf{n} \cdot (\mathbf{g} - \mathbf{g}^*), L_j)_{\partial \mathsf{D}^k} \ , \tag{3.127b}$$

$$\mathbf{P}^k(p) = (\nabla p, L_j)_{\mathsf{D}^k} - (\mathbf{n}(p - p^*), L_j)_{\partial \mathsf{D}^k} \ , \tag{3.127c}$$

to obtain the DG formulation

$$\left(\frac{\partial \mathbf{u}}{\partial t}, L_j\right)_{\mathsf{D}^k} + N^k(\mathbf{u}) = -\mathbf{P}^k(p) + \frac{1}{Re}\frac{\nu}{\rho} L^k(\mathbf{g}) - (\mathbf{u} - \mathbf{u}^*, L_j)_{\partial \Omega} \tag{3.128a}$$

$$(\nabla u_i, L_j)_{\mathsf{D}^k} = (\mathbf{g}(u_i), L_j)_{\mathsf{D}^k} + (\mathbf{n}(u_i - u_i^*), L_j)_{\partial \mathsf{D}^k} \tag{3.128b}$$

where the $(\cdot,\cdot)_{\partial\Omega}$ account for applying the global boundary conditions. Defining discrete versions of the above

$$\mathsf{N}^k(\mathbf{u}_i) = \mathsf{S}^k\ (\mathbf{u}\mathbf{u}_i) - \mathsf{F}^k(\mathbf{n} \cdot (\mathbf{u}\mathbf{u}_i - (\mathbf{u}\mathbf{u}_i)^*)) \tag{3.129a}$$

$$\mathsf{L}^k(\mathbf{g}) = \mathsf{S}^k\ \mathbf{g} - \mathsf{F}^k(\mathbf{n} \cdot (\mathbf{g} - \mathbf{g}^*)) \tag{3.129b}$$

$$\mathsf{P}^k(\mathbf{p}) = \mathsf{S}^k\ \mathbf{p} - \mathsf{F}^k(\mathbf{n}(\mathbf{p} - \mathbf{p}^*)) \tag{3.129c}$$

we end up with the following locally defined nodal scheme

$$\mathsf{M}^k \frac{\partial \mathbf{u}}{\partial t} + \mathsf{N}^k(\mathbf{u}) = -\mathsf{P}^k(\mathbf{p}) + \frac{1}{Re}\frac{\nu}{\rho}\mathsf{L}^k(\mathbf{g}) - \mathsf{F}^k_{\partial\Omega}(\mathbf{u} - \mathbf{u}^*_{\partial\Omega}), \tag{3.130a}$$

$$\mathsf{S}^k \mathbf{u} = \mathsf{M}^k \mathbf{g} + \mathsf{F}^k(\mathbf{n}(\mathbf{u}_i - \mathbf{u}_i^*)), \tag{3.130b}$$

where again $\mathbf{g}$ can be eliminated locally.

3.6 Final Remarks on the DG Method

The DG method was originally developed for conservation laws. It has been used for flow simulations and the compressible Navier Stokes equations, e.g. in [6, 33, 73]. Solving the incompressible Navier Stokes (INS) equations using high order spectral schemes, then usually continuous finite element/spectral element methods have been applied, as in [61, 72]. There are not much reported work on the DG method for the unsteady incompressible Navier Stokes equations. A few found: [45] using a vorticity stream-function formulation, while [13] considers stationary flow.

The presentation here on the discontinuous Galerkin for conservation laws are well known and well described. Though, to my knowledge, there is no work describing the entire process, from the discontinuous Galerkin formulation, the application of high order discontinuous spectral elements, efficient creation of operators, and implementation details, as presented here.

3.6.1 Symmetric Elliptic Operator

In Section 3.4 we considered the DG discretization of the elliptic problem

$$\nabla \cdot \frac{1}{\rho} \nabla p = f \; . \tag{3.131}$$

There we split the operator into two first order equations. We have indicated that the resulting discretized operator is not symmetric, though having the property of $N(\mathcal{L})$ being spanned by the constant vector. We have tried to make the operator symmetric, but using the LDG technique, we have not had success. In finite element methods, the approach goes like

$$\int_{\mathsf{D}} (\nabla \cdot \frac{1}{\rho} \nabla p) v \; \mathrm{d}\mathbf{x} = \int_{\mathsf{D}} f \; v \; \mathrm{d}\mathbf{x} \; , \tag{3.132}$$

then by integration by parts

$$-\int_{\Omega} (\frac{1}{\rho} \nabla p) \cdot \nabla v \; \mathrm{d}\mathbf{x} + \int_{\partial\Omega} (\mathbf{n} \cdot \frac{1}{\rho} \nabla p) v \; \mathrm{d}\mathbf{x} = \int_{\Omega} f \; v \; \mathrm{d}\mathbf{x} \; . \tag{3.133}$$

Inserting L_j for v and expanding p and f in L_i we get

$$\begin{aligned} -\int_{\Omega} (\frac{1}{\rho} \nabla \sum_i p_i L_i) \cdot \nabla L_j \; \mathrm{d}\mathbf{x} + \int_{\partial\Omega} (\mathbf{n} \cdot \frac{1}{\rho} \nabla \sum_i p_i L_i) L_j \; \mathrm{d}\mathbf{x} \\ = \int_{\Omega} \sum_i f_i L_i \; L_j \; \mathrm{d}\mathbf{x} \; . \end{aligned} \tag{3.134}$$

After some manipulations we get in matrix form

$$-\mathsf{L} \; \mathbf{p} = \mathsf{M} \; \mathbf{f} - \mathsf{F} \; \mathbf{p}_{BC} \tag{3.135}$$

where

$$(\mathsf{L})_{ij} = \int_{\Omega} \frac{1}{\rho} \nabla L_j \cdot \nabla L_i \; \mathrm{d}\mathbf{x} \; , \tag{3.136a}$$

$$(\mathsf{M})_{ij} = \int_{\Omega} L_i \; L_j \; \mathrm{d}\mathbf{x} \; , \tag{3.136b}$$

$$(\mathsf{F})_{ij} = \int_{\partial\Omega} (\mathbf{n} \cdot \frac{1}{\rho} \nabla L_i) L_j \; \mathrm{d}\mathbf{x} \; . \tag{3.136c}$$

The boundary conditions, namely the $\mathsf{F} \; \mathbf{p}_{BC}$ part, goes to the right-hand side, and the L is symmetric. Hence if using a continuous spectral element or finite element method, we should ensure the operator to be symmetric for every kind of boundary conditions. This would simplify the solution process, since now a symmetric solver can be used.

It would be interesting to see whether this continuous system would work hand in hand with the discontinuous one. We need to find a way to evaluate the integral (3.136a) efficiently, since the operator must be recomputed for every time step.

C H A P T E R 4

Temporal Scheme

Time is the soul of this world
— Pythagoras ~582–507 BC

We have now described how to discretize in space, the spatial scheme, using the discontinuous Galerkin method. Now time has come to describe how to integrate the equations forward in time, the temporal scheme.

The incompressible Navier Stokes (INS) equations (2.1) are a system of equations combining non-linear advection, linear diffusion, and an algebraic equation. On top of the INS, we shall need to solve the level set equation (2.15).

Solving a linear diffusion problem of the form

$$\frac{\partial u}{\partial t} = \frac{1}{Re}\frac{\partial^2 u}{\partial x^2} \tag{4.1}$$

using an explicit scheme, may lead to severe time step restrictions on the temporal scheme. The reason comes from the growing maximum absolute eigenvalue of the diffusion operator. Consider the equation

$$\frac{\partial u}{\partial t} = -\lambda u, \qquad u(0) = u_0; \tag{4.2}$$

having the solution $y(t) = \exp(-\lambda t)$. The solution for $\Re(\lambda) > 0$ approaches zero exponentially in time. Hence should also every solution of a temporal scheme approach zero as $t \to \infty$. This is the traditional linear stability criterion for a temporal scheme [28, 40]. Using an explicit Euler scheme for Equation (4.2),

$$u_{i+1} = u_i - \lambda\Delta t u_i = (1 - \lambda\Delta t)u_i \ , \tag{4.3}$$

allows us to integrate forward in time, $u_i = u(i\Delta t)$,

$$u_1 = (1 - \lambda\Delta t)u_0 \tag{4.4a}$$

$$u_2 = (1 - \lambda\Delta t)u_1 = (1 - \lambda\Delta t)^2 u_0 \tag{4.4b}$$

$$u_n = (1 - \lambda\Delta t)^n u_0 \ , \tag{4.4c}$$

which will approach zero when $|1 - \lambda\Delta t| < 1$. For a real positive λ, we get

$$\lambda\Delta t < 2\,. \tag{4.5}$$

Hence the explicit Euler scheme is stable when $\lambda\Delta t < 2$. We must therefore choose Δt small enough to fulfill Equation (4.5). For other explicit schemes, the condition (4.5) will not be the exact same, but they all have a stability restriction of the form $\lambda\Delta t < a$ for some $a > 0$, hence for increasing λ, Δt must be decreased.

If we make a similar eigenvalue analysis of the diffusion problem (4.1), again using an explicit Euler approximation of the time derivative, we will obtain similar results, i.e. the time step will be limited by the inverse of the largest absolute eigenvalue. For a typical second order approximation of the diffusion operator using finite difference or finite element approximation, we mentioned in Section 3.4 that the eigenvalues grow as $\mathcal{O}(1/(Re\ \Delta x^2))$, while for a spectral element method they grow as $\mathcal{O}(\mathsf{P}^4/(Re\ \Delta x^2))$, where P is the polynomial order of the elements. Thus, the time step must be similar decreased to order $\mathcal{O}(Re\ \Delta x^2/\mathsf{P}^4)$ in order to obtain a stable method. For low Reynolds numbers Re, one would have to take a huge amount of time steps. A common solution is to use an implicit method. An example is the implicit Euler method for the diffusion problem

$$(\mathsf{I} - \Delta t \frac{1}{Re}\mathsf{D})u_{i+1} = \mathsf{T}u_{i+1} = u_i\ , \tag{4.6}$$

where D is a discrete diffusion operator. This requires a linear solve for the operator $\mathsf{T} = (\mathsf{I} - \Delta t \frac{1}{Re}\mathsf{D})$ in each time step. For a diffusion operator, we would prefer an implicit method to overcome the time step restriction.

The INS is non-linear. Using an implicit scheme for a non-linear system, the linear solve is replaced by a non-linear solve. A non-linear solver is usually based on a Newton type iteration, where each iteration requires a linear solve. Thus, in each time step, to solve the non-linear problem, we would need a number of linear solves. On contrary would an explicit method for the advection term work well: It will not give such a strict time step restriction. Hence in this case would an implicit method result in extra work without actually improving the result. Thus we would prefer an explicit method for the non-linear term to avoid the iterative Newton process.

A common solution when solving the INS is therefore to use an explicit-implicit splitting, treating the diffusion part implicitly and the nonlinear part explicitly.

Furthermore the INS contains an algebraic equation, which the velocities must fulfill, Equation (2.1b). The task is to find a pressure field p, which, inserted in Equation (2.1a), will produce a velocity field satisfying the algebraic equation (2.1b). In general this requires solving a system for both the velocities and the pressure simultaneously. It is a linear solve, though, it is known to be difficult and time consuming to solve.

We would therefore like to decouple the calculation of the pressure from that of the velocities. This sounds attractive, but it is not straightforward. The splitting and decoupling may introduce errors, termed time splitting errors, if the splitting and decoupling is not applied with care. Examples are [8, 9, 26, 27] [Chorin,Kim&Moin] . They all show second order accuracy on the velocity variables, and some also obtain second order on the pressure.

When including the level set equation in the system, we have yet another equation which in terms of computational cost should add as little overhead to the solution method as possible. Thus, we would like to decouple the evaluation of the level set equation from the remaining system.

With this in mind, we are looking for methods that have a high degree of flexibility in terms of splitting and decoupling. We are also looking for higher order methods, hoping that the higher order methods can provide more accurate solutions for less computational work.

We shall in this section first describe a semi-implicit deferred correction method. The advantages of this method is its flexibility and will be evident later. Then we shall describe some of the problems encountered, when solving a differential and algebraic equation simultaneously, and how these problems can be solved. We will perform a small test of the semi-implicit deferred correction method, which will give an idea of the types of differential algebraic systems, where the method is applicable. Finally, we shall show how to apply these approaches to the INS equations.

4.1 Semi-Implicit Spectral Deferred Correction

A deferred correction method consist of two basic steps. The first step calculates a preliminary solution. The next step calculates a correction to the preliminary solution. The calculation of the correction is deferred until the preliminary solution is set. Adding the correction to the preliminary solution gives a more accurate solution. Which can be considered as a new preliminary solution and another correction step may be computed, thereby iteratively improving the accuracy of the solution.

We shall now present a deferred correction approach, using spectral techniques in the deferred correction step, and furthermore allowing explicit-implicit splitting of the system, thus the name semi-implicit spectral deferred correction (SISDC).

The method has our interest due to its semi-implicit nature and high order of accuracy. It has been used for solving the incompressible Navier Stokes equations, e.g., in [48], where the method is shown to integrate the incompressible Navier Stokes equations in time to arbitrary order of accuracy.

The SISDC method is based on low order time integration methods for the computation of the preliminary solution and the correction. The time splitting errors that may arise, are corrected as any other part of the error, hence the accuracy of the final solution does not suffer errors due to the splitting. We can split and decouple terms as needed – an attractive and flexible feature.

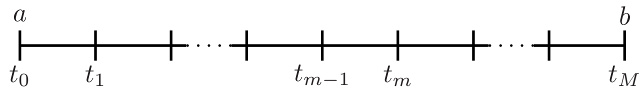
We will in this section describe the SISDC for an ODE, and later we will apply the method to the INS.

4.1.1 SISDC for ODEs

Let us first introduce the SISDC for ODEs for which we shall follow the original development in [19]. Assume we want to solve the ODE

$$u'(t) = f\big(u(t), t\big), \qquad u(a) = u_a, \qquad t \in [a, b] \; . \tag{4.7}$$

Before we continue, some definitions are appropriate. Let t_i be a grid in the interval, $a = t_0 < t_1 < ... < t_M = b$, and let $\mathbf{u} = (u_0, ..., u_M)$ be the values of $u(t)$ at the



nodes, $u_i = u(t_i)$. Define the Mth order Lagrange interpolant as

$$\tilde{u}(t) = L_M(\mathbf{u}, t) = \sum_{i=0}^{M} u_i \; \pi_i(t), \tag{4.8}$$

where the fundamental polynomials $\pi_i(t)$ are defined by

$$\pi_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{M} \frac{t - t_j}{t_i - t_j}. \tag{4.9}$$

The Lagrange interpolant (4.8) is a continuous function, hence we can differentiate and integrate it.

We define the integration operator I_j^{j+1} as

$$\mathsf{I}_j^{j+1}\mathbf{u} = \int_{t_j}^{t_{j+1}} \tilde{u}(\tau) \; \mathrm{d}\tau, \quad j = 0, ..., M-1 \; . \tag{4.10}$$

The integration operator integrate $\tilde{u}(t)$ between t_j and t_{j+1}, i.e. the gray part in Figure 4.1. We evaluate this integral numerically using a quadrature rule on the

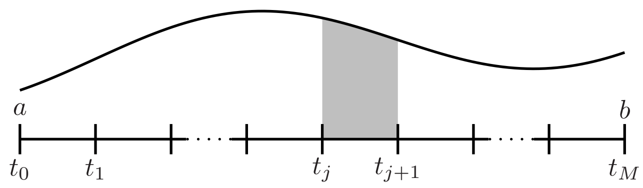


Figure 4.1:

form

$$\mathsf{I}_j^{j+1}\mathbf{u} = \sum_{i=0}^{M} \alpha_{ji} u_i \; , \tag{4.11}$$

hence we may think of l_j^{j+1} as a vector and the evaluation of the integral as an inner product of l_j^{j+1} and the vector $\mathbf{u}$. Let us analyze how accurate this quadrature rule can be: We have $M+1$ nodes in between a and b, represented in the vector $\mathbf{u}$. The position of the $M+1$ nodes are set, but we have $M+1$ weights which we can choose freely. Hence we can determine the $M+1$ weights in order to integrate a polynomial of degree at most M exactly. Note that none of the nodes in the quadrature is actually inside the integration interval. The Lagrange interpolant $\tilde{u}$ in Equation (4.8) is a polynomial of degree M, hence it can integrated exactly.

Notice that there are M integrations of this type, and thereby M quadrature rules, one for each interval. All the interval integrals can be determined by a matrix-vector product, the l_j^{j+1} being the $(j+1)$'th row in the matrix.

The grid $\{t_i\}$ is typically a Chebyshev grid or similar used in spectral methods, hence the Lagrange interpolant (4.8) and above integration has spectral accuracy.

Returning to the SISDC, the spectral deferred correction method is based on the Picard integral form of the ODE (4.7)

$$u(t) = \int_a^t f\big(u(\tau), \tau\big)\ \mathrm{d}\tau + u_a\ . \tag{4.12}$$

Given an approximation of the solution $v(t)$, let the error be $\delta(t)$, hence $u(t) = v(t) + \delta(t)$. Substituting $u(t)$ in the Integral form (4.12) gives

$$v(t) + \delta(t) = \int_a^t f\big(v(\tau) + \delta(\tau), \tau\big)\ \mathrm{d}\tau + u_a. \tag{4.13}$$

This is also a Picard form of an ODE, hence we could actually solve for the error $\delta(t)$, and use the result to update the approximation v. If we want p'th order accuracy, we would have to evaluate the integral to p'th order accuracy, i.e. using a p'th order time integration scheme. But if we can evaluate this equation to p'th order, we could just as well evaluate the original Picard form (4.12) to p'th order, hence we would have p'th order accuracy already, and we would not be needing this equation.

Continue as follows: Inserting $v(t)$ for $u(t)$ in Equation (4.12) will not hold, but the remainder can be considered as a kind of error indicator. Hence a measure of the error is the residual equation

$$\epsilon(t) = \int_a^t f\big(v(\tau), \tau\big)\ \mathrm{d}\tau + u_a - v(t). \tag{4.14}$$

Subtracting Eq. (4.14) from Eq. (4.13) gives a correction equation

$$\delta(t) = \int_a^t \Big(f\big(v(\tau) + \delta(\tau), \tau\big) - f\big(v(\tau), \tau\big)\Big)\ \mathrm{d}\tau + \epsilon(t)\ , \tag{4.15}$$

which is also on the same form as (4.12), a Picard integral form of an ODE. This equation tells how well the error measure $\epsilon(t)$ approximates the true error $\delta(t)$.

This correction equation we can solve in the same manner as we solved the original ODE, and the result can be used for updating the solution. However, we do not need a p'th order scheme for the integral in Equation (4.15).

The argument goes as follows: Let $h = b - a$ be the interval length. Assume $v(t)$ is order r accurate, i.e. $u(t) = v(t) + \mathcal{O}(h^r)$. Then $\delta(t) = \mathcal{O}(h^r)$. The magnitude of $\epsilon(t)$ can be found by considering the residual equation (4.14), assuming for notational simplicity that f does not depend on time

$$\begin{aligned}
\epsilon(t) &= \int_a^t f\big(v(\tau)\big)\ \mathrm{d}\tau + u_a - v(t) \\
&= \int_a^t f\big(u(\tau) - \delta(\tau)\big)\ \mathrm{d}\tau + u_a - v(t) \\
&\approx \int_a^t \Big(f\big(u(\tau)\big) - f'\big(u(\tau)\big)\delta(\tau)\Big)\ \mathrm{d}\tau + u_a - v(t) \\
&= u(t) - v(t) - \int_a^t f'\big(u(\tau)\big)\delta(\tau)\ \mathrm{d}\tau \\
&= \delta(t) - \int_a^t f'\big(u(\tau)\big)\delta(\tau)\ \mathrm{d}\tau \\
&= \mathcal{O}(h^r)\ . \qquad (4.16)
\end{aligned}$$

where we have Taylor expanded f at $u(\tau)$. We have here assumed f to be sufficiently smooth, such that $f'(u)$ is bounded. Consider the integral in Equation (4.15), call it $\epsilon_\delta(t)$

$$\begin{aligned}
\epsilon_\delta(t) &= \int_a^t \Big(f\big(v(\tau) + \delta(\tau)\big) - f\big(v(\tau)\big)\Big)\ \mathrm{d}\tau \\
&\approx \int_a^t \Big(f\big(v(\tau)\big) + f'\big(v(\tau)\big)\delta(\tau) - f\big(v(\tau)\big)\Big)\ \mathrm{d}\tau \\
&= \int_a^t f'\big(v(\tau)\big)\delta(\tau)\ \mathrm{d}\tau = \mathcal{O}(h^r) \qquad (4.17)
\end{aligned}$$

Assume that we evaluate the residual equation (4.14) exactly, getting the exact $\epsilon(t)$. If we also evaluate $\epsilon_\delta(t)$ exactly, we would get the exact error $\delta(t)$ and the exact solution $u(t)$,

$$u(t) = v(t) + \delta(t) = v(t) + \epsilon_\delta(t) + \epsilon(t)\ . \qquad (4.18)$$

If we evaluate $\epsilon_\delta(t)$ using a order s time integration scheme, then we would only get a $\mathcal{O}(h^s)$ correct result, i.e. $\tilde{\epsilon}_\delta(t) = \epsilon_\delta(1 + \mathcal{O}(h^s))$. Using this to update our solution to a new approximation, i.e. $v^*(t) = v(t) + \delta(t)$, we would get

$$\begin{aligned}
v^*(t) &= v(t) + \epsilon_\delta(t)(1 + \mathcal{O}(h^s)) + \epsilon(t) \\
&= u(t) + \epsilon_\delta(t)\mathcal{O}(h^s) = u(t) + \mathcal{O}(h^{r+s})\ , \qquad (4.19)
\end{aligned}$$

hence the new solution $v^*(t)$ will be r orders more accurate than the original approximation $v(t)$.

In practice, we cannot evaluate the residual equation (4.14) exactly. But if we just evaluate it to order $\mathcal{O}(h^{r+s})$, then that would be sufficient, not adding lower order errors to $v^*(t)$ in Equation (4.19). The residual equation integrates $f(v(\tau))$, but since the approximate solution $v(t)$ is known, we do not need a time integration scheme as for the integral in the correction equation (4.15). Any standard $\mathcal{O}(h^{r+s})$ quadrature rule may be used.

The articles found on spectral deferred correction [19, 49] does not include any description on how the method works, like the one presented here. They only state how to evaluate the correction equation. The presentation should give an idea of, why the method is formulated that way, and how it works.

We will now apply this setup to the i'th interval in the grid, $h_i = t_{i+1} - t_i$. Consider first Equation (4.12) for the i'th interval, using an explicit Euler approximation for the integral we get the approximation

$$v_{i+1} = h_i f(v_i, t_i) + u_i. \tag{4.20}$$

Similarly, using an explicit Euler approximation for the correction equation (4.15) gives

$$\delta_{i+1} = \delta_i + h_i \left(f(v_i + \delta_i, t_i) - f(v_i, t_i) \right) + \epsilon_{i+1} - \epsilon_i. \tag{4.21}$$

Subtract the residual equation (4.14) at time t_i from that at time t_{i+1}, to get

$$\epsilon_{i+1} - \epsilon_i = \int_{t_i}^{t_{i+1}} f(v(\tau), \tau)\ \mathrm{d}\tau - v_{i+1} + v_i. \tag{4.22}$$

Combining with (4.21) gives a direct equation for the updated solution $v^*_{i+1} = v_{i+1} + \delta_{i+1}$

$$v^*_{i+1} = v^*_i + h_i \left(f(v^*_i, t_i) - f(v_i, t_i) \right) + \int_{t_i}^{t_{i+1}} f(v(\tau), \tau)\ \mathrm{d}\tau. \tag{4.23}$$

To complete the correction procedure, we need to specify how to calculate the integral in (4.23). This is where we need an accurate evaluation, which we base on the quadrature rule (4.11). Let $f_i = f(v_i, t_i)$, and let $\tilde{f}$ be the Lagrange interpolant of the f_i's. We can now integrate using (4.10)

$$\int_{t_i}^{t_{i+1}} f(v(\tau), \tau)\ \mathrm{d}\tau \approx \int_{t_i}^{t_{i+1}} \tilde{f}\ \mathrm{d}\tau = \mathsf{I}_i^{i+1}\ \mathbf{f}, \tag{4.24}$$

which completes the scheme. A similar expression exists for the implicit Euler approximation

$$v^*_{i+1} = v^*_i + h_i \left(f(v^*_{i+1}, t_{i+1}) - f(v_{i+1}, t_{i+1}) \right) + \mathsf{I}_i^{i+1}\ \mathbf{f}\ . \tag{4.25}$$

In [49] the method above is extended in a semi-implicit manner, where the right hand side f is split into different parts treated explicitly and implicitly in a straight forward manner.

One can repeat the process with another correction step, using the new v_i^* as v_i, which will raise the order of the approximation by one each time. This can continue as long as the integral (4.24) is evaluated sufficiently accurately. Using $M+1$ points in the Lagrange interpolant provides an error of size $O(h^{M+1})$ for the integral, and an $O(h^{M+1})$ global accuracy for the solution. A complete error analysis is provided in [19].

4.2 Differential Algebraic Equations

An ordinary differential equation (ODE) is often given on the form

$$\dot{y} = f(y) \tag{4.26}$$

where y may be a vector and f a vector function. Given some initial condition $y(0) = y_0$, this is solved by stepping the solution forward in time, using some time integration scheme.

A differential algebraic equation (DAE) has besides a number of differential equations also a number of algebraic equations,

$$\dot{y} = f(y, z), \tag{4.27a}$$
$$0 = g(y, z). \tag{4.27b}$$

The time-stepping procedures from the world of ODE's are not directly applicable to a system of this kind, since they do not give a way of progressing the algebraic variable z in time.

The Incompressible Navier Stokes equations (2.1) can be written in the same form as Equations (4.27). The intention of this section is to explore theory from the world of Differential Algebraic Equations (DAE's), putting attention on the form in which the incompressible Navier-Stokes equations are written.

4.2.1 Index and Index Reduction

The solution method of the DAE system (4.27) depends on what is called the index of the DAE.

The most simple case is when $g_z(y, z)$ is invertible in a neighborhood of the solution. Then the "Implicit Function Theorem" states that the algebraic equations possess a locally unique solution of the form $z = G(y)$. Substituting into the differential equation (4.27a), we get

$$\dot{y} = f(y, G(y)), \tag{4.28}$$

which is a standard ODE, which we know how to solve. If $g_z(y, z)$ is not invertible, we have to find another procedure.

Differentiate the algebraic equation (4.27b) with respect to time t, and using the chain-rule and Equation (4.27a) we get

$$0 = g_y \dot{y} + g_z \dot{z}, \qquad \Leftrightarrow \quad \dot{z} = \frac{g_y}{g_z}\dot{y} = \frac{g_y}{g_z} f, \tag{4.29}$$

suddenly the algebraic equations is turned into an ODE for z, (4.29) and (4.27a) together forms a system of ODE's. This system is called the underlying system of ODE's.

Definition 4.2.1 (Differentiation Index). Differentiation index is the minimum number of differentiations of the algebraic equations needed to be able to extract an explicit system of ODE's.

According to the definition, a system of ODE's has index 0. The DAE system (4.27) has index one, since one differentiation and some manipulation give us a system of ODE's.

If however g_z is not invertible, then $\dot{z}$ can not be extracted as in (4.29). Consider e.g.

$$\dot{y} = f(y, z), \tag{4.30a}$$
$$0 = g(y), \tag{4.30b}$$

differentiating (4.30b) ones gives

$$0 = g_y \dot{y}, \qquad 0 = g_y(y) f(y, z), \tag{4.31}$$

but g_z is zero and not invertible. However, the algebraic equations is now on the form (4.27b). Differentiating once more

$$0 = g_{yy} f^2 + g_y(f_y f + f_z \dot{z}). \tag{4.32}$$

Assuming now $g_y f_z$ is invertible then

$$\dot{z} = -\frac{1}{g_y f_z}(g_{yy} f^2 + g_y f_y f), \tag{4.33}$$

hence we have found the underlying ODE system. The original system is an index 2 system.

The procedure of differentiating to lower the index of the system is called *index reduction.*

Example 4.1: The pendulum

Consider the undamped pendulum in Figure 4.2. It has two degrees of freedom, and in Lagrangian coordinates the equation of motion is given by the ODE

$$\dot{\theta} = \psi \tag{4.34a}$$
$$\dot{\psi} = -\frac{g}{l}\sin(\theta). \tag{4.34b}$$

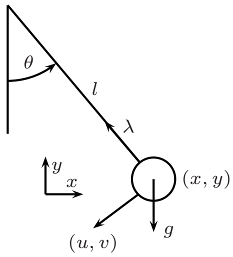


Figure 4.2: Pendulum

In Eulerian coordinates we shall use the length of the (mass-less) string l, the mass m, gravity g, and the force on the mass from the string λ. We have

$$x = l\sin(\theta) \qquad\qquad y = -l\cos(\theta)\ ,$$

and using Newtons second law $ma = F$ gives

$$\begin{aligned} m\dot{u} &= -\lambda\sin(\theta) & m\dot{v} &= \lambda\cos(\theta) - mg \\ \dot{u} &= -\frac{\lambda}{ml}x & \dot{v} &= -\frac{\lambda}{lm}y - g \end{aligned}$$

In Eulerian coordinates the system is described by the differential algebraic system

$$\dot{x} = u, \tag{4.35a}$$

$$\dot{y} = v, \tag{4.35b}$$

$$\dot{u} = -\frac{1}{lm}x\lambda, \tag{4.35c}$$

$$\dot{v} = -\frac{1}{lm}y\lambda - g, \tag{4.35d}$$

$$0 = x^2 + y^2 - l^2\ , \tag{4.35e}$$

The system gives no ODE for the string force λ. Successive differentiation and manipulation of the algebraic constraint (4.35e) gives first

$$0 = 2xu + 2yv \quad \Leftrightarrow \quad 0 = xu + yv\ . \tag{4.36}$$

The second differentiation:

$$\begin{aligned} 0 &= \dot{x}u + x\dot{u} + \dot{y}v + y\dot{v} \\ &= u^2 - \frac{1}{lm}x^2\lambda + v^2 - \frac{1}{lm}y^2\lambda - yg \\ &= u^2 + v^2 - \frac{l}{m}\lambda - yg\ , \end{aligned} \tag{4.37}$$

using $l^2 = x^2 + y^2$. And finally

$$\begin{aligned} 0 &= 2u\dot{u} + 2v\dot{v} - \frac{l}{m}\dot{\lambda} - \dot{y}g, \\ &= -\frac{2}{lm}ux\lambda - \frac{2}{lm}vy\lambda - 2vg - \frac{l}{m}\dot{\lambda} - vg, \\ &= -\frac{2}{lm}\lambda\underbrace{(ux + vy)}_{=0} - \frac{l}{m}\dot{\lambda} - 3vg \end{aligned} \tag{4.38}$$

giving a differential equation for the force λ,

$$\dot{\lambda} = -\frac{3m}{l} vg \; . \tag{4.39}$$

Three differentiations was used to make a system of ODE's, so the pendulum system is at most index 3. At most, since in general a procedure like this only shows that it is possible using 3 differentiations, but more clever manipulations may have given a system of ODE's using only 2 differentiations. This system is however an index 3 system.

Note the meaning of the different algebraic equations:

- The "zero order" algebraic constraint (4.35e) sets the length of the string.
- The "first order" algebraic constraint (4.36) states that the velocity is orthogonal to the string
- The "second order" algebraic constraint (4.37) is a kind of energy balance equation.

End of Example 4.1.

4.2.2 Consistent Initial Conditions

An ODE is solvable for any initial conditions. This is not the case for a DAE. For instance when giving initial conditions to a system like the pendulum (4.35), the initial condition must fulfill the algebraic equation (4.35e), saying that the mass is at the end of the string. Initial conditions must however also fulfill all "hidden" algebraic constraints as (4.36) and (4.37). When all constraints are fulfilled, the initial conditions are called consistent.

4.2.3 Drift-off Phenomena

If applying an ODE solver to the underlying system of ODE's given consistent initial conditions, due to numerical errors the solution will drift away from the algebraic constraints. This is not a problem for the ODE solver, since the underlying ODE system has a solution for every initial condition, also the inconsistent in the DAE sense.

For the pendulum, the solution and error in the 3 algebraic equations are shown in Figure 4.3 for a simple explicit Euler time-integration of Equations (4.35a-d) and (4.39). As time goes, the error grows and the method is actually unstable - the error grows in some polynomial or exponential-like way.

A solution to the drift off is once in a while to project the solution back on the subspaces (manifolds) defined by the algebraic constraints. This projections should be done often enough to avoid instability, and in a way not destroying accuracy properties of the ODE solver. It is usually not necessary to project on all the constraints, see [28] for examples.

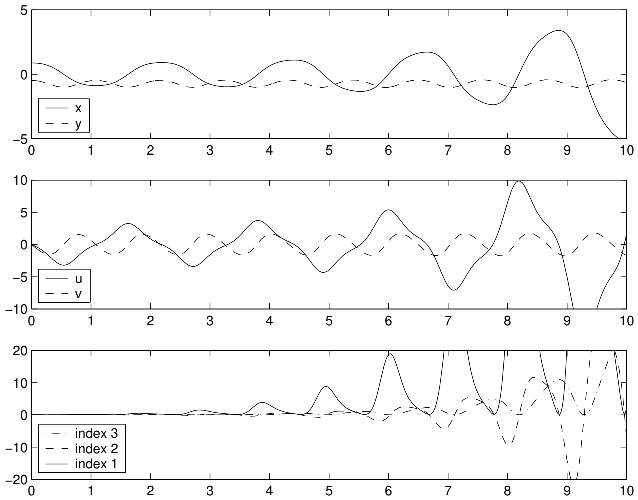


Figure 4.3: Top, middle: Solution. Bottom: Error in the algebraic constraints

4.2.4 Projection

Let $\tilde{\mathbf{u}}_n$ come from a numerical method, and assume it does not fulfill the algebraic equations. We want a consistent solution, i.e. one which is on the manifold described by the algebraic equations, so we want to project $\tilde{\mathbf{u}}_n$ onto this manifold, giving a $\mathbf{u}_n$ which is as close to $\tilde{\mathbf{u}}_n$ as possible. Solving the, in general, nonlinear minimization problem

$$\min_{g(\mathbf{u}_n)=0} ||\mathbf{u}_n - \tilde{\mathbf{u}}_n||_2 \tag{4.40}$$

will produce the desired solution.

It is shown in [20], that if solving the underlying ODE's for a DAE on the form in Equations (4.27)[1] with any one-step procedure as e.g. Runge Kutta methods or a BDF multi-step procedure, and applying a projection procedure as above at the end of each step, then the solution method for the DAE will have the same order as the original method used on an ODE.

Projection for Linear Algebraic Equations

Assume the algebraic equations are linear in $\mathbf{u}$, i.e. can be written on the form

$$\mathbf{0} = \mathbf{g}(\mathbf{u}) = \mathsf{H}\mathbf{u} + \mathbf{z}(t). \tag{4.41}$$

[1]In DAE terms, the form in Equations (4.27) is called semi explicit

Notice that the equations might be nonlinear in t. Then the projection can be obtained using the *Moore-Penrose-Pseudo-Inverse*[2]

$$\mathsf{H}^{+} = \mathsf{H}^{T}(\mathsf{H}\mathsf{H}^{T})^{-1}. \tag{4.42}$$

Let

$$\mathsf{P} = \mathsf{I} - \mathsf{H}^{+}\mathsf{H} = \mathsf{I} - \mathsf{H}^{T}\left(\mathsf{H}\mathsf{H}^{T}\right)^{-1}\mathsf{H}. \tag{4.43}$$

Then

$$\mathbf{u}_n = -\mathsf{H}^{+}\mathbf{z}_n + \mathsf{P}\tilde{\mathbf{u}}_n. \tag{4.44}$$

See also: [20]. This projection technique will prove useful later.

4.2.5 Runge Kutta Methods

Instead of solving the underlying ODE and then project, the entire system of differential and algebraic equations can be evaluated simultaneously using e.g. a Runge Kutta method.

A general Runge Kutta method with s stages can be written on the form

$$Y_{ni} = y_n + h\sum_{j=1}^{s} a_{ij}\dot{Y}_{nj}, \tag{4.45a}$$

$$\dot{Y}_{ni} = f(Y_{ni}), \quad i = 1..s \tag{4.45b}$$

$$y_{n+1} = y_n + h\sum_{j=1}^{s} b_j\dot{Y}_{nj} \tag{4.45c}$$

where the two first equations must be solved simultaneously for each $i = 1, ..., s$. Afterwards the last equation is evaluated to give the solution. For DAE's this can be generalized to

$$\begin{aligned} Y_{ni} &= y_n + h\sum_{j=1}^{s} a_{ij}\dot{Y}_{nj} & Z_{ni} &= z_n + h\sum_{j=1}^{s} a_{ij}\dot{Z}_{nj} \\ \dot{Y}_{ni} &= f(Y_{ni}, Z_{ni}) & 0 &= g(Y_{ni}, Z_{ni}), \quad i = 1..s \end{aligned} \tag{4.46a}$$

$$y_{n+1} = y_n + h\sum_{j=1}^{s} b_j\dot{Y}_{nj} \qquad z_{n+1} = z_n + h\sum_{j=1}^{s} b_j\dot{Z}_{nj}, \tag{4.46b}$$

which result in a (nonlinear) system of equations to solve simultaneously for Y_{ni}, $\dot{Y}_{ni}$, Z_{ni}, and $\dot{Z}_{ni}$ for each stage $i = 1, ..., s$. Notice the "artificial" $\dot{Z}_{nj}$ which is needed to update the final solution. This procedure implies that the solution at each stage fulfills the algebraic constraint, but the solution (y_{n+1}, z_{n+1}), being a

[2]This definition is valid only when $(\mathsf{H}\mathsf{H}^{T})^{-1}$ exists.

linear combination of the stage solutions, does in general not fulfill the algebraic constraints. It is however close enough.

We shall here only consider SDIRK[3] methods, $a_{ij} = 0$ for $j > i$ and $a_{ii} = a_{jj}$ for $i, j > 1$. We shall also require that $a_{s,i} = b_i$, which simplifies the procedure to

$$Y_{ni} = y_n + h \sum_{j=1}^{i} a_{i,j} f(Y_{nj}, Z_{nj}), \tag{4.47a}$$

$$0 = g(Y_{ni}, Z_{ni}), \quad i = 1..s \tag{4.47b}$$

$$y_{n+1} = Y_{ns}, \qquad z_{n+1} = Z_{ns} \tag{4.47c}$$

Next, we state a number of theorems from [29] for Runge Kutta methods, where p will be the classical order of the Runge Kutta method, and q will be the stage order, often denoted $C(q)$. Stage order is the order of accuracy of the stages Y_{ni}, which is usually less than the classical order.

Theorem 3.1 For index 1 systems, if $b_i = a_{si}$, then

$$z_n - z(x_n) = O(h^p). \tag{4.48}$$

Hence the order of accuracy for all variables, including the algebraic, will be the same as the classical order.

Theorem 4.1 States existence and uniqueness for the DAE index 2 system when solved with a Runge-Kutta method.

Lemma 4.3+Theorem 4.4+Theorem 4.6 For an index 2 system with $p \geq q + 1$ and $q \geq 1$

$$y_n - y(x_n) = O(h^{q+1}) \tag{4.49a}$$

$$z_n - z(x_n) = O(h^q) \tag{4.49b}$$

hence the order of accuracy of the method is limited by the stage order of the Runge Kutta method.

Theorem 6.4 Index 3 system with $p \geq q + 1$ and $q \geq 2$

$$y_n - y(x_n) = O(h^q) \tag{4.50a}$$

$$z_n - z(x_n) = O(h^{q-1}) \tag{4.50b}$$

hence the order of accuracy of the method is again limited by the stage order of the Runge Kutta method, which must be at least 2 in order to work at all.

The INS equations being an index 2 system, it is comforting to see that there exist a unique solution. A drawback is however that to achieve high order, we

[3]Singly Diagonally Implicit Runge Kutta

would have to use Runge Kutta methods with high stage order. Higher stage order in Runge Kutta methods is achieved by solving for several stages simultaneously. In a INS setup, solving for several stages at the same time is computationally very expensive. Hence the Runge Kutta approach is not optimal.

4.2.6 Preliminary Numerical Test of DAE Solvers

We have performed preliminary test on a number of different DAE solvers, of which we will present two here. The test have been conducted on different versions of the pendulum equations. The two first are based on the linear approximation.

$$u_1' = u_2, \tag{4.51a}$$
$$u_2' = -u_1, \qquad \mathbf{u}(0) = (1,0),\ t = [0; 2\pi + 0.2]\ , \tag{4.51b}$$

simulating a bit longer than one period, while the last is the true nonlinear pendulum equation from Example 4.1.

1) linear pendulum, linear extension, index 2

$$\begin{aligned} u_1' &= u_2 \\ u_2' &= -u_1 + u_3 + u_4 \\ u_3' &= -u_1 + u_2 \\ 0 &= u_1 + u_2 - u_3 \end{aligned}$$

with initial conditions $\mathbf{u} = (1; 0; 1; -1)$, having the solution

$$\begin{aligned} u_1 &= \cos(t) \\ u_2 &= -\sin(t) \\ u_3 &= \cos(t) - \sin(t) \\ u_4 &= -\cos(t) + \sin(t). \end{aligned}$$

2) linear pendulum, non-linear extension, index 2

$$\begin{aligned} u_1' &= u_2 \\ u_2' &= -u_1 + u_3^2 + u_4 \\ u_3' &= 2u_1u_2 - u_1 \\ 0 &= u_1^2 + u_2 - u_3 \end{aligned}$$

with initial conditions $\mathbf{u} = (1; 0; 1; -1)$, having the solution

$$\begin{aligned} u_1 &= \cos(t) \\ u_2 &= -\sin(t) \\ u_3 &= 0.5\cos(2t) - \sin(t) + 0.5 \\ u_4 &= -(0.5\cos(2t) - \sin(t) + 0.5)^2 \end{aligned}$$

3) non-linear pendulum, index 3

$$
\begin{aligned}
u_1' =& u_3 \\
u_2' =& u_4 \\
u_3' =& -u_1 u_5 \\
u_4' =& -u_2 u_5 - g \\
0 =& u_1^2 + u_2^2 - l^2
\end{aligned}
$$

using $m = 1$ and $l = 1$ and initial conditions $\mathbf{u} = (\cos(-0.5); \sin(-0.5); 0; 0, \lambda)$, choosing λ such that the system is consistent. A reference solution is calculated by solving the Lagrangian equations (4.34) to high precision.

Runge Kutta Method – GERK 3

The first method we will test, is a Runge Kutta method. We shall use the procedure described in section 4.2.5, to iterate the implicit equation together with the algebraic equations for each implicit stage. The Runge Kutta we will consider, is called GERK 3, and is an SDIRK method of order 3, with embedded 4th order error estimator, stage order $C(2)$, and A-stable with $R(-\infty) = 0$, i.e., L-stable, presented in [68]. For a definition on A- and L-stability, consult for example [28, 40]. The Runge Kutta-tableau for GERK 3:

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{5}{6} & \frac{5}{12} & \frac{5}{12} & & \\
\frac{10}{21} & \frac{95}{588} & \frac{-5}{48} & \frac{5}{12} & \\
1 & \frac{59}{600} & \frac{-31}{75} & \frac{539}{600} & \frac{5}{12} \\
\hline
 & \frac{59}{600} & \frac{-31}{75} & \frac{539}{600} & \frac{5}{12} \\
\mathrm{e} & \frac{55}{600} & \frac{55}{75} & \frac{-245}{600} & \frac{-5}{12}
\end{array}
$$

We shall present only the order results for the GERK 3 method, i.e., the numbers here are the order of accuracy achieved. Order plots are found in Appendix B.

Index 2				Index 3	
linear		nonlinear		pendulum	
x	z	x	z	x	z
3	3	3	2	2	1

This complies with the theorems stated in last section on Runge Kutta methods for DAE systems.

Spectral Deferred Correction

We shall also test the implicit spectral deferred correction (ISDC) procedure, the fully implicit version of [49]. The update equations is:

$$\begin{aligned} y^*_{i+1} &= y^*_i + h_i \left(f(y^*_{i+1}, z^*_{i+1}) - f(y_{i+1}, z_{i+1}) \right) + \mathsf{I}_i^{i+1}\ \mathbf{f}\ , && (4.57a)\\ 0 &= g(y^*_{i+1})\ . && (4.57b) \end{aligned}$$

As for the Runge Kutta method, we shall only present order results for different orders of the ISDC method, order plots are found in Appendix B:

ISDC order 2

Index 2				Index 3	
linear		nonlinear		pendulum	
x	z	x	z	x	z
2	2	2	1	1	1

ISDC order 3

Index 2				Index 3	
linear		nonlinear		pendulum	
x	z	x	z	x	z
3	3	3	1	1	0

ISDC order 4

Index 2				Index 3	
linear		nonlinear		pendulum	
x	z	x	z	x	z
4	4	4	1	1	0

ISDC order 5

Index 2				Index 3	
linear		nonlinear		pendulum	
x	z	x	z	x	z
5	5	5	1	1	0

ISDC order 6

Index 2				Index 3	
linear		nonlinear		pendulum	
x	z	x	z	x	z
6	6	6	1	1	0

We have not found any reported results testing the spectral deferred correction method for DAE systems. There are several interesting questions arising:

For the nonlinear index 2 problem, even though the algebraic variable is only 1st order, all differential variables remain full order. What actually happens is unknown. Since the differential variables have high order, then it is most likely possible to retract high order on the algebraic variables also. The question is what is needed to get the high order in the algebraic variable, if possible.

The method does not work for a nonlinear DAE of index 3. Runge Kutta methods should have at least stage order 2 in order to work for an index 3 system. We may argue, that a deferred correction approach based on first order Euler method only has what corresponds to stage order 1, and if using a second order method, we would have what corresponds to stage order 2. We have not further pursued what will happen then, hence we do not know whether it will work, or if any other approach would work for the index 3 system.

The comforting news is that the method shows full order for the linear index 2 problem. That is promising, since the incompressible Navier Stokes equations are linear in the algebraic equation. If linearity for the algebraic equation is the only condition, then it should be possible to get full order also for the Navier Stokes equations. If the nonlinear part in the momentum equation are important, we should still get full order for the differential variables. Then we must reconsider the problem of how to get high order also for the algebraic variables.

4.3 Incompressible Navier Stokes

So far we have described the semi implicit spectral deferred correction method, and introduced some theory on differential algebraic equations. We will now combine this theory and apply it to the incompressible Navier Stokes equations. First we will make a connection to the DAE theory, and then we will describe different projection approaches.

The incompressible Navier-Stokes Equations:

$$\frac{\partial u_i}{\partial t} + \nabla \cdot (u_i \mathbf{u}) = \nabla \cdot (\mu \nabla u_i) - \frac{1}{\rho}\frac{\partial p}{\partial x_i} + g_i, \quad i = 1, \ldots, d, \tag{4.58a}$$

$$0 = \nabla \cdot \mathbf{u}, \tag{4.58b}$$

can be written on the semi-explicit index 2 form, ignoring density and viscosity

$$\frac{\partial \mathbf{u}}{\partial t} = f(\mathbf{u}, p), \tag{4.59a}$$

$$0 = g(\mathbf{u}). \tag{4.59b}$$

4.3.1 Existence of Solution for INS-equations

According to [29], an index 2 problem must fulfill

$$||\left(g_{\mathbf{u}}(\mathbf{u}) f_z(\mathbf{u}, z)\right)^{-1}|| \leq M \tag{4.60}$$

in a neighborhood of the exact solution. Otherwise it is not an index 2 problem but a higher index problem. In case of (4.60) a unique solution exists.

For the incompressible NS we have

$$g_{\mathbf{u}}(\mathbf{u}) = [\partial_x, \partial_y, \partial_z], \tag{4.61}$$

$$f_p(\mathbf{u}, p) = [\partial_x, \partial_y, \partial_z]^T, \tag{4.62}$$

hence

$$g_{\mathbf{u}}(\mathbf{u}) f_z(\mathbf{u}, z) = \nabla^2, \tag{4.63}$$

i.e. a Laplace operator, must be invertible and the inverse must be bounded.

Theorem 4.3.1 (Bounded Inverse Theorem). *Let $T : V \to W$ be a bounded, linear and bijective operator from the Banach space V onto the Banach space W. Then the inverse operator $T^{-1} : W \to V$ is also bounded.*

This is Theorem 5.13 of [56], where also a proof of the theorem can be found.

Consider the requirement for the incompressible NS, the Laplace problem (4.63): It is linear. It is injective and surjective (there exist a solution and the solution is unique),[4] hence bijective. Furthermore, in an appropriate chosen norm, differentiation can be shown to be bounded, see e.g. Example 3.1 of [56]. Hence the Bounded inverse theorem applies to the Laplace problem (4.63), and there exists a unique solution to the incompressible NS.

4.3.2 The Underlying ODE and Projection

Differentiate the algebraic constraint (4.58b), defining $f(\mathbf{u}, p) = F(\mathbf{u}) - \nabla p$, gives

$$\begin{aligned} \frac{d}{dt} g(\mathbf{u}) &= \nabla \cdot \dot{\mathbf{u}} = \nabla \cdot f(\mathbf{u}, p) \\ &= \nabla \cdot F(\mathbf{u}) - \nabla^2 p = h(\mathbf{u}, p). \end{aligned} \tag{4.64}$$

Differentiating once again gives

$$\begin{aligned} \frac{d}{dt} h(\mathbf{u}, p) &= \frac{\partial h(\mathbf{u}, p)}{\partial \mathbf{u}} \dot{\mathbf{u}} + \frac{\partial h(\mathbf{u}, p)}{\partial p} \dot{p} \\ &= \nabla \cdot F'(\mathbf{u}) f(\mathbf{u}, p) - \nabla^2 \dot{p} \end{aligned} \tag{4.65}$$

hence

$$\nabla^2 \dot{p} = \nabla \cdot F'(\mathbf{u})(F(\mathbf{u}) - \nabla p). \tag{4.66}$$

So, solving the underlying ODE involves solving a Poisson problem. This is tricky, since pressure boundary condition is in general unknown.

This system of underlying ODE's will also suffer from drift off. Hence a projection to the divergence free manifold $g(\mathbf{u}) = 0$ is necessary. Following the projection procedure (4.41)-(4.44), in our case $\mathbf{z}(t) = 0$, $\mathsf{H} = \nabla$,

$$\mathbf{u}_n = \mathsf{P} \tilde{\mathbf{u}}_n \,, \tag{4.67}$$

where again $\tilde{\mathbf{u}}_n$ is coming from some numerical integrator and

$$\mathsf{P} = \mathsf{I} - \nabla \left(\nabla^2\right)^{-1} \nabla \,. \tag{4.68}$$

We have not found anyone using this approach for solving the INS. It may be due to the following reason: Firstly we need to solve a Poisson equation for each time step. Secondly we need to solve a Poisson equation each time we wish to

[4] For the Neumann Problem the solution is unique up to a constant, but that is sufficient.

project to the divergence free manifold. The Poisson solve is an expensive part from a computational point of view. As we shall see in the next section, we can do with one Poisson solve per time step, hence this approach may be computationally too expensive. In both cases we do in general not know the pressure boundary conditions for the Poisson problem, and they need to be approximated from the flow.

4.3.3 Fractional Step/Projection Methods

This section describes a common approach of solving the Navier-Stokes equations. It goes in two steps (fractions) and use a projection step, hence their name. Consider a simple forward Euler approximation of the time derivative in the Navier-Stokes equation.

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t \left(\mathbf{F}(\mathbf{u}_n) - \nabla p_{n+1} \right), \tag{4.69a}$$

$$0 = \nabla \cdot \mathbf{u}_{n+1}. \tag{4.69b}$$

Assume for simplicity that $\nabla \cdot \mathbf{u}_n = 0$. From start, ∇p_{n+1} is not known, but the idea is to find a ∇p_{n+1} such that $\nabla \cdot \mathbf{u}_{n+1} = 0$. Take the divergence of the momentum equation (4.69a) to obtain

$$\nabla \cdot \mathbf{u}_{n+1} = \nabla \cdot \mathbf{u}_n + \Delta t \left(\nabla \cdot \mathbf{F}(\mathbf{u}_n) - \nabla^2 p_{n+1} \right) \tag{4.70}$$

We want $\nabla \cdot \mathbf{u}_{n+1} = 0$ and we have[5] $\nabla \cdot \mathbf{u}_n = 0$, hence

$$0 = \nabla \cdot \mathbf{F}(\mathbf{u}_n) - \nabla^2 p_{n+1} \tag{4.71}$$

This is exactly the time derivative of the algebraic constraint, Equation (4.64). Hence we require the solution to fulfill the index 1 algebraic constraint, but not the original index 2 algebraic constraint. But as mentioned before, it is not always necessary to project onto all the algebraic constraints.

The fractional step procedure goes in two steps:

- Insert $\mathbf{u}_n$ in (4.71) and solve for p_{n+1}.
- Use p_{n+1} in (4.69a) to obtain $\mathbf{u}_{n+1}$. Now $\mathbf{u}_{n+1}$ will automatically be divergence free.

This fractional step method is equivalent to a formulation using the projection operator (4.43). Consider a pressure free approximation

$$\tilde{\mathbf{u}}_{n+1} = \mathbf{u}_n + \Delta t\, \mathbf{F}(\mathbf{u}_n), \tag{4.72}$$

[5] Otherwise just keep the term, and it will be a term in Equation (4.71)

and use the projection operator (4.43) to get back onto the divergence free manifold

$$\begin{aligned}\mathbf{u}_{n+1} &= \left(\mathsf{I} - \nabla \left(\nabla^2\right)^{-1} \nabla\right) \tilde{\mathbf{u}}_{n+1} \\ &= \left(\mathsf{I} - \nabla \left(\nabla^2\right)^{-1} \nabla\right) \left(\mathbf{u}_n + \Delta t\, \mathbf{F}(\mathbf{u}_n)\right) \\ &= \mathbf{u}_n + \Delta t\, \mathbf{F}(\mathbf{u}_n) - \nabla \left(\nabla^2\right)^{-1} \nabla \cdot \left(\mathbf{u}_n + \Delta t\, \mathbf{F}(\mathbf{u}_n)\right) \\ &= \mathbf{u}_n + \Delta t\, \mathbf{F}(\mathbf{u}_n) - \Delta t \nabla \underbrace{\left(\nabla^2\right)^{-1} \nabla \cdot \mathbf{F}(\mathbf{u}_n)}_{p_{n+1}}.\end{aligned} \tag{4.73}$$

This is exactly the fractional step procedure.

A better approximation is to use the last known value of the pressure, that is

$$\tilde{\mathbf{u}}_{n+1} = \mathbf{u}_n + \Delta t \left(\mathbf{F}(\mathbf{u}_n) - \nabla p_n\right) \tag{4.74}$$

Actually, we seek to determine

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t \left(\mathbf{F}(\mathbf{u}_n) - \nabla p_{n+1}\right) \ . \tag{4.75}$$

Assume we want to update the velocities on the form

$$\mathbf{u}_{n+1} = \tilde{\mathbf{u}}_{n+1} - \Delta t \nabla \phi_n \tag{4.76}$$

Use Equation (4.76) to eliminate $\tilde{\mathbf{u}}$ from Equation (4.74)

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t \left(\mathbf{F}(\mathbf{u}_n) - \nabla p_n - \nabla \phi_n\right) \ . \tag{4.77}$$

Matching with Equation (4.75), we get an update for the pressure,

$$\nabla p_{n+1} = \nabla p_n + \nabla \phi. \tag{4.78}$$

The ϕ can be found by taking the divergence of Equation (4.76),

$$\Delta t \nabla^2 \phi_n = \nabla \cdot \tilde{\mathbf{u}}_{n+1}, \tag{4.79}$$

using that $\nabla \mathbf{u}_{n+1} = 0$. This procedure is again similar to using the projection operator P and using $\mathbf{u}_{n+1} = \mathsf{P}\tilde{\mathbf{u}}$.

Notice that Section 4.2.4 promised that if solving the underlying ODE's and applying a projection procedure, then the solution method would retain order properties for the DAE. Here we do not solve the underlying ODE, so even though the same projection approach is used, we cannot say anything about the accuracy of the method beforehand.

4.3.4 Pressure Projection

The pressure projection approach is very similar to the fractional step method. The main difference is that the Pressure Projection uses a semi-implicit Euler approximation. The technique can be used also for other time integration schemes. Consider

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h\big(-\nabla p_{n+1} + \mathcal{N}(\mathbf{u}_n) + \mathcal{L}(\mathbf{u}_{n+1}) + \mathbf{f}_{n+1}\big), \tag{4.80a}$$
$$0 = \nabla \cdot \mathbf{u}_{n+1} \ , \tag{4.80b}$$

with $\mathcal{N}(\mathbf{u})$ being the nonlinear advection and $\mathcal{L}(\mathbf{u})$ the diffusion. The purpose of the pressure variable is to assure no divergence in the new velocities. The pressure projection starts by computing a preliminary $\bar{\mathbf{u}}_{n+1}$ using the old pressure

$$\bar{\mathbf{u}}_{n+1} - h\mathcal{L}(\bar{\mathbf{u}}_{n+1}) = \mathbf{u}_n + h\big(-\nabla p_n + \mathcal{N}(\mathbf{u}_n) + \mathbf{f}_{n+1}\big), \tag{4.81}$$

The bar above the velocity variable $\bar{\mathbf{u}}_{n+1}$ indicates that the variable is not divergence free. Then we perform the projection onto a divergence free velocity field

$$\left.\begin{aligned} \mathbf{u}_{n+1} &= \bar{\mathbf{u}}_{n+1} - h\nabla\phi, \\ \nabla\cdot\mathbf{u}_{n+1} &= 0, \end{aligned}\right\} \quad \rightarrow \quad \nabla^2\phi = \frac{1}{h}\nabla\cdot\bar{\mathbf{u}}_{n+1}. \tag{4.82}$$

Substituting $\bar{\mathbf{u}}_{n+1}$ in (4.81) using (4.82) gives

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h\big(-\nabla p_n - \nabla\phi + h\mathcal{L}(\nabla\phi) + \mathcal{L}(\mathbf{u}_{n+1}) + \mathcal{N}(\mathbf{u}_n) + \mathbf{f}_{n+1}\big) \tag{4.83}$$

Matching the original momentum equation (4.80a) gives a pressure update

$$p_{n+1} = p_n + \phi - h\mathcal{L}(\phi). \tag{4.84}$$

The update matches Equation (4.78), apart from the last term. The last term becomes important especially for higher order methods. If not included, then lower order errors are introduced, see e.g. [9].

4.3.5 Velocity Projection

We will now illustrate the standard velocity projection technique using a semi-implicit Euler approximation. We consider the following

$$\frac{\partial\mathbf{u}}{\partial t} = -\frac{1}{\rho}\nabla p + \mathcal{N}(\mathbf{u}) + \mathcal{L}(\mathbf{u}) + \mathbf{f}, \quad \mathbf{u}|_{\partial\Omega} = \mathbf{u}_b \tag{4.85a}$$
$$\nabla\cdot\mathbf{u} = 0, \tag{4.85b}$$

The purpose of the pressure variable is to assure no divergence in the new velocities. Consider a semi-implicit approximation of the time derivative, removing the implicit diffusive part

$$\mathbf{u}_{n+1} = \bar{\mathbf{u}}_n + h\big(-\frac{1}{\rho}\nabla p_{n+1} + \mathcal{N}(\bar{\mathbf{u}}_n) + \mathbf{f}_{n+1}\big), \tag{4.86a}$$
$$0 = \nabla\cdot\mathbf{u}_{n+1}, \tag{4.86b}$$

giving the following equation for the pressure

$$\nabla \cdot \frac{1}{\rho} \nabla p_{n+1} = \nabla \cdot \big(\frac{1}{h} \bar{\mathbf{u}}_n + \mathcal{N}(\bar{\mathbf{u}}_n) + \mathbf{f}_{n+1} \big). \tag{4.87}$$

This is a Poisson problem, and Neumann boundary conditions can be found from the original equation

$$\mathbf{n} \cdot \nabla p = \mathbf{n} \cdot \rho \left(-\frac{\partial \mathbf{u}}{\partial t} + \mathcal{N}(\mathbf{u}) + \mathcal{L}(\mathbf{u}) + \mathbf{f} \right), \quad \text{on } \partial\Omega \tag{4.88}$$

which must be approximated to same order as the time derivative. For higher order method, this implies extrapolation of the non-linear as well as the linear term [27, 42]. Having the pressure, we can insert it into (4.86a) and as the last step we solve for the new velocities.

$$\bar{\mathbf{u}}_{n+1} - h\mathcal{L}(\bar{\mathbf{u}}_{n+1}) = \mathbf{u}_{n+1} \tag{4.89}$$

The velocities will not produce completely divergence free variables in the end, however, it is sufficiently divergence free.

Velocity projection is often used with an Adams-Moulton or BDF-type approximation of the time derivative and extrapolation of the nonlinear part, hence a k'th order BDF(k) method:

$$\frac{1}{h} \sum_{i=-1}^{k-1} \alpha_i \mathbf{u}_{n-i} = -\nabla p_{n+1} + \sum_{i=0}^{k} \beta_i \mathcal{N}(\mathbf{u}_{n-i}) + \mathcal{L}(\mathbf{u}_{n+1}) + \mathbf{f}_{n+1}, \tag{4.90}$$

where the α_i's are the BDF coefficients and the β_i's are extrapolation coefficients. This formulation is 1st order using a BDF(1) method, 2nd order using a BDF(2), and gives $O((\Delta t)^{5/2})$ order using a BDF(3). For reference, see [26, 27, 42]

The velocity projection approach may yield $O((\Delta t)^{5/2})$, but being able to discretize in space to basically any order, on would like to mimic this in space. Furthermore, the BDF method in its original form is fully implicit, making it difficult to include more equations, e.g. the level set equation, in the solution process without increasing the amount of work beyond reason, or by extensive use of extrapolation like for the non-linear term above. Finally, how to apply Neumann type of boundary conditions for the velocities and retain the order is not known [26, 27].

4.3.6 SISDC for INS

We will now apply the SISDC method from Section 4.1.1 to the INS. This is also presented in [48], where the alternative formulation $\mathbf{u} = \mathbf{m} + \nabla\chi$ is used, thereby eliminating the pressure variable. This alternative formulation, however, makes it difficult to apply velocity boundary conditions and to obtain and use the pressure, which is required in the free surface flow formulation. We have developed a variant, that uses the velocity and pressure variables directly, which we shall present in the following.

Consider the integral form of the momentum equation:

$$\mathbf{u}(t_{i+1}, x) = \mathbf{u}(t_i, x) + \int_{t_i}^{t_{i+1}} \left(-\nabla p + \mathcal{N}(\mathbf{u}) + \mathcal{L}(\mathbf{u}) + \mathbf{f}\right)\, \mathrm{d}t \tag{4.91}$$

We want to remove the pressure from the spectral integration. Evaluate the pressure part of the integral using the mean value theorem,

$$\mathbf{u}(t_{i+1}, x) = \mathbf{u}(t_i, x) - \Delta t_i \nabla p(\xi_i) + \int_{t_i}^{t_{i+1}} \left(\mathcal{N}(\mathbf{u}) + \mathcal{L}(\mathbf{u}) + \mathbf{f}\right)\, \mathrm{d}t \tag{4.92}$$

where ξ_i is unknown, $t_i < \xi_i < t_{i+1}$. Now we can construct the SISDC method, treating the linear term implicitly and the nonlinear explicitly, to obtain

$$\begin{aligned} \mathbf{u}^*_{i+1} = \mathbf{u}^*_i + \Delta t \big[&-\nabla p^*_{\xi_i} + \mathcal{N}(\mathbf{u}^*_i) - \mathcal{N}(\mathbf{u}_i) + \\ &\mathcal{L}(\mathbf{u}^*_{i+1}) - \mathcal{L}(\mathbf{u}_{i+1}) \big] + \mathsf{I}_i^{i+1} F(\mathbf{u}), && (4.93a) \\ 0 = \nabla \cdot \mathbf{u}^*_{i+1}. & && (4.93b) \end{aligned}$$

where $F(\mathbf{u})$ is the integrand of Equation (4.92). This formulation can be combined with either a pressure projection or velocity projection approach.

Our first attempts using the SISDC on the INS was without the application of the mean value theorem on the pressure part. That resulted in a system of equations on the form

$$\begin{aligned} \mathbf{u}^*_{i+1} = \mathbf{u}^*_i + \Delta t \big[&-\nabla p^*_{i+1} + \nabla p_{i+1} + \mathcal{N}(\mathbf{u}^*_i) - \mathcal{N}(\mathbf{u}_i) + \\ &\mathcal{L}(\mathbf{u}^*_{i+1}) - \mathcal{L}(\mathbf{u}_{i+1}) \big] + \mathsf{I}_i^{i+1} F(\mathbf{u}, p), && (4.94a) \\ 0 = \nabla \cdot \mathbf{u}^*_{i+1}. & && (4.94b) \end{aligned}$$

We did however not have success with this form, but we have on the other hand not found any specific reason for why it should not work.

4.3.7 Test of SISDC on INS

We shall here present a test of the velocity projection SISDC method on the following traveling wave solution of the incompressible Navier Stokes equations

$$\begin{aligned} u &= \frac{3}{4} + \frac{1}{4}\cos(2\pi(x-t))\sin(2\pi(y-t))e^{-\alpha 8\pi^2 t}, && (4.95a) \\ v &= \frac{3}{4} - \frac{1}{4}\sin(2\pi(x-t))\cos(2\pi(y-t))e^{-\alpha 8\pi^2 t}, && (4.95b) \\ p &= \frac{1}{64}\big(\cos(4\pi(x-t)) + \cos(4\pi(y-t))\big)e^{-\alpha 16\pi^2 t}. && (4.95c) \end{aligned}$$

The α determines the decay of the waves. Setting $\alpha = 0$ means that the waves travel undisturbed in time. The right hand side forcing $\mathbf{f}$ in the momentum equation is found by inserting the solution into the Navier Stokes equation. The solution is periodic, hence it is possible to set it up in a $[0;1] \times [0;1]$ box using periodic boundary conditions. The solution is calculated up till time $t = 4$.

We set up the problem on a grid having 226 elements of order 4. Figure 4.4 contains error plots for the test problem set up using periodic boundary conditions. The velocities all show more or less optimal order. The 4th order method seems

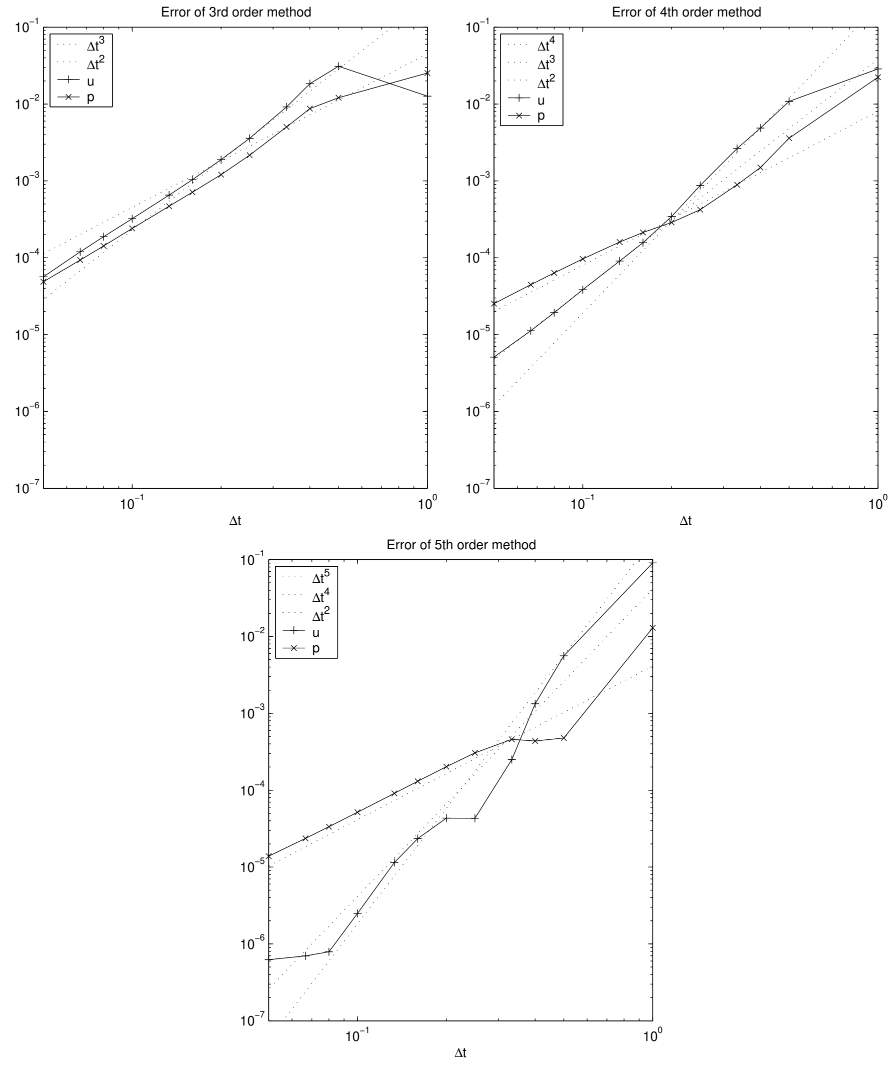


Figure 4.4: Order Results 3rd to 5th order

to have some kind of order reduction, reducing to 3rd order for small time steps. However the 5th order method does not show alike. The 5th order method hits the limit of the spatial discretization at around 10^{-6} and does not improve beyond this point.

The pressure shows only second order in all plots. We do not know the pressure time $t_{i+\xi}$, hence the pressure is compared with the exact solution at time $t_{i+1/2}$, which is only second order accurate. However since the velocities are high order, we expect the pressure to be of equally high order.

4.3.8 Boundary Conditions in SISDC

So far we have said nothing about boundary conditions. While boundary conditions for the velocities usually are available, conditions for the pressure are not and these must be approximated from the flow. As mentioned before, it is well known, that the Neumann condition for the pressure needs to be approximated to the same order in time as the time stepping method to avoid order reduction, see e.g. [27, 42].

Applying velocity conditions to the SISDC method is not straight forward. Imposing prescribed boundary conditions on the implicit step even for simple problems like the heat equations will cause an order reduction similar to that of Runge Kutta methods, i.e., it is caused by the low stage-order, which is determined by the order of the internal scheme in the SISDC scheme. Using a high-order internal scheme in the SISDC may help alleviate this problem, although no analysis is available. Further discussions along this line can be found in [48].

Furthermore, it is not known how to approximate the pressure Neumann condition to sufficient order in a useful way. So far we have only been able to obtain 1st order accuracy for the pressure, limiting the order of accuracy for the velocities to 2nd order for the SISDC method with complex boundary conditions.

4.3.9 Time Stepping INS and the Level Set Equation

The level set equation, as the nonlinear part of the Navier-Stokes equation, has no need for implicit time stepping. Hence we want to include it in the system in a way requiring as little extra work as possible. e.g. in the form

$$\begin{aligned}
\mathbf{u}_{i+1} &= \mathbf{F}(\mathbf{u}_{i+1},\, \mathbf{u}_i,\, p_{i+1},\, \phi_i),\\
\nabla\cdot\mathbf{u}_{i+1} &= 0,\\
\phi_{i+1} &= \mathbf{G}(\mathbf{u}_{i+1},\, \mathbf{u}_i,\, \phi_i),
\end{aligned}$$

where subscript $i+1$ refers implicit and i explicit dependency. We solve for the velocities and the pressure as before, with the only difference that viscosity and density may vary in time. Afterwards we solve the level set equation, using the newly calculated velocity $\mathbf{u}^{i+1}$.

The above formulation fits directly into the SISDC projection method, using explicit or implicit approximations as appropriate. Unfortunately, as discussed above, it remains unknown how to apply pressure boundary conditions without impacting the accuracy adversely.

For problems requiring such types of boundary conditions, e.g., non-periodic problems, we may consider other methods more appropriate and we shall use the classic BDF-type scheme described previously.

In the equation for the velocities, we need the level set at time $t+1$, hence we need to extrapolate, which we do with an explicit Adams method of same order as the BDF method. When the pressure and the velocities are calculated, we will calculate the level set explicitly based on a BDF scheme

$$\frac{1}{h}\sum_{i=-1}^{k-1} \alpha_i \phi_{n-i} = -\mathbf{u}^{n+1} \cdot \nabla \bar{\phi}_{n+1} \tag{4.96}$$

where $\bar{\phi}_{n+1}$ is the extrapolated level set, and $\mathbf{u}^{n+1}$ the newly calculated velocity. Applying the BDF scheme for the level set, and simply using the value from the Adams method, minimizes the time splitting errors.

CHAPTER 5

Level Set Modeling of the Free Surface

By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work.

— Johann Von Neumann 1903–1957

We shall describe the use of level set techniques for the modeling of free surfaces.

Level set methods are fairly new, introduced in 1988 in [54]. Since then, level set methods have gained popularity in many fields of science, including, among others, computer vision and graphics [21], image restoration and enhancement, shape recognition and recovery [46], tracking of discontinuities in e.g. hyperbolic conservation laws [4], and as we shall present, free surface flow modeling. A fairly comprehensive book on the topic, including many relevent references, is [53].

We introduced the level set function in Chapter 2. That was in the continuous case, where the choice of the level set function is arbitrary, as long as it is differentiable and fulfills Equation (2.6). In the discrete case it is from a numerical point of view advantageous to require the level set function to have a number of properties.

5.1 Signed Distance Functions

We will choose the level set function ϕ to be the signed distance to the interface. Figure 5.1 shows a 1D version of a level set function where the interface is a point, and the distance to the point will be the straight line through the point having slope one. Figure 5.2 shows a 2D version of a level set function where the interface is a circle at center $(0, 0)$ and radius 0.3. The level set function ϕ everywhere has the shortest signed distance to the circle, hence ϕ is a cone.

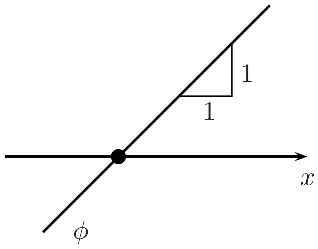


Figure 5.1: A level set function in 1D

Definition 5.1.1 (distance function). let Γ represent the interface between the two fluids, then the distance function is defined as

$$d(\mathbf{x}) = \min_{\mathbf{x}\in\Omega}(|\mathbf{x} - \mathbf{x}_\Gamma|), \quad \forall\, \mathbf{x}_\Gamma \in \Gamma \tag{5.1}$$

Call the point in Γ minimizing above for $\mathbf{x}_\Gamma^{\min}(\mathbf{x})$

Definition 5.1.2 (signed distance function). Let Ω^+ and Ω^- define the outside and the inside of an interface. The signed distance is defined as

$$\phi(\mathbf{x}) = \begin{cases} d(\mathbf{x}) & \forall\, x \in \Omega^+ \\ -d(\mathbf{x}) & \forall\, x \in \Omega^- \end{cases} \tag{5.2}$$

Using a signed distance function to represent the interface Γ implies that the interface is defined implicitly by the zero contour of the level set, $\Gamma = \{\mathbf{x} \mid \phi(\mathbf{x}) = 0\}$. Given an explicit representation of the interface, it is not trivial in general to find the distance function, i.e. to carry out the minimization in Equation (5.1).

A distance function has a number of properties. The size of the gradient is one,

$$|\nabla\phi| = 1, \tag{5.3}$$

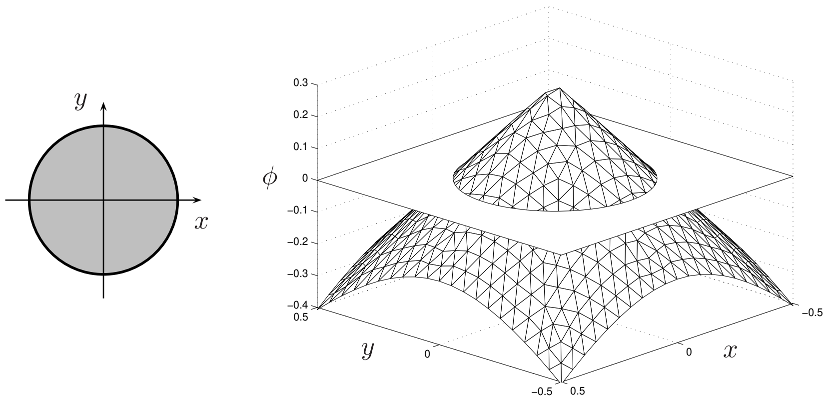


Figure 5.2: A droplet represented by a level set function in 2D

almost everywhere. Except where $\mathbf{x}_\Gamma^{\min}(\mathbf{x})$ is not unique, meaning that the point $\mathbf{x}$ is equidistant to at least two points on Γ. An example is the center of the circle in Figure 5.2, $\mathbf{x} = (0, 0)$, which is equidistant to the entire interface.

The unit normal of the interface is determined by

$$\mathbf{n}_\Gamma = \left.\frac{\nabla\phi}{|\nabla\phi|}\right|_{\phi=0} \tag{5.4}$$

and we can theoretically find the closest point on the interface from

$$\mathbf{x}_\Gamma^{\min}(\mathbf{x}) = \mathbf{x} - \phi(\mathbf{x})\mathbf{w}(\mathbf{x}), \quad \mathbf{w}(\mathbf{x}) = \frac{\nabla\phi(\mathbf{x})}{|\nabla\phi(\mathbf{x})|} \tag{5.5}$$

since the shortest distance from $\mathbf{x}$ to the interface is the straight line perpendicular to the interface at $\mathbf{x}_\Gamma^{\min}(\mathbf{x})$, and the gradient at $\mathbf{x}$ will take the direction of the interface normal at $\mathbf{x}_\Gamma^{\min}(\mathbf{x})$, a side effect of ϕ being a signed distance. This is again not possible everywhere, i.e. when $\mathbf{x}$ is equidistant to at least two points on Γ and $\nabla\phi(\mathbf{x})$ is not defined.

The mean curvature of the interface can be defined as

$$\kappa = \nabla \cdot \left.\frac{\nabla\phi}{|\nabla\phi|}\right|_{\phi=0} \tag{5.6}$$

Since for a distance function $|\nabla\phi| = 1$, above could be simplified. However we may in certain cases allow the level set function to only be approximately a signed distance function, hence e.g. Equation (5.5) may not be exact, and it is appropriate to keep the normalization constant in the presentation here.

5.1.1 Interface Thickness

To avoid handling discontinuities in e.g. the viscosity $\mu(\phi)$ and density $\rho(\phi)$, the interface is given a thickness of size ϵ, i.e., we use a smoothed Heaviside function

$$H(\phi, \epsilon) = \begin{cases} 0 & \phi \quad < -\epsilon \\ \frac{1}{2} + \frac{\phi}{2\epsilon} + \frac{1}{2\pi}\sin\left(\frac{\pi\phi}{\epsilon}\right) & -\epsilon \quad \le \phi \le \epsilon \\ 1 & \phi \quad > \epsilon \end{cases} \tag{5.7}$$

The smoothed Heaviside function is depicted in Figure 5.3. This is only one of many possible definitions of a smoothed Heaviside function [69]. Using a high order method to enable high precision, it is important that the function is as smooth as possible. The above Heaviside function has continuous first and second derivative, while the third is discontinuous at $\phi = \pm\epsilon$. It is possible to design higher order polynomials or combinations of polynomials and trigonometric functions which are as smooth as required.

The interface thickness ϵ should be chosen as small as possible for accuracy but large enough to stabilize the system: If chosen too small, the Heaviside function

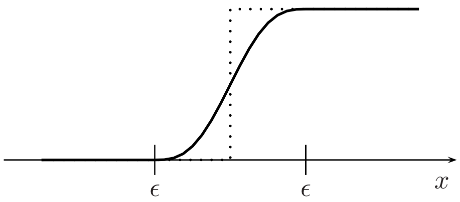


Figure 5.3: Smoothed Heaviside function

will be too steep, hence it becomes difficult to represent and oscillations due to Runge phenomenon may occur, making the system stiff and difficult to solve. A typical choice is ϵ of the size of an element. Note that ϵ does not have to be a global constant, but may vary throughout the domain.

Example 5.1: Smoothed Heaviside and δ-functions
For the Heaviside function, determine the polynomial $p(x)$ of degree $N = 2k + 1$

$$p(x) = \sum_{j=0}^{N} a_j x^j \tag{5.8}$$

which fulfills

$$0 = p(-1), \tag{5.9a}$$
$$1 = p(1), \tag{5.9b}$$
$$0 = \left.\frac{\partial^i p(x)}{\partial x^i}\right|_{x=-1}, \quad i = 1, ..., k \tag{5.9c}$$
$$0 = \left.\frac{\partial^i p(x)}{\partial x^i}\right|_{x=1}, \quad i = 1, ..., k. \tag{5.9d}$$

Inserting (5.8) we get $2k + 2$ equations and equally many unknowns. Then define

$$H(\phi, \epsilon) = \begin{cases} 0 & \phi \quad < -\epsilon \\ p\left(\frac{\phi}{\epsilon}\right) & -\epsilon \quad \leq \phi \leq \epsilon \\ 1 & \phi \quad > \epsilon \end{cases} \tag{5.10}$$

which will have k continuous derivatives everywhere. Figure 5.4 shows 5 different smoothed Heaviside functions of different polynomial degrees. The polynomial with highest degree is the most smooth but has also the steepest gradient at $\phi = 0$. Hence the coefficients and the size of the derivatives grow larger with the degree of smoothness, which also makes the differentiation less accurate.

When a fully smooth function is needed, and the thickness is less important,

$$H_s(\phi, \epsilon) = \frac{1}{2} + \frac{\phi}{2\sqrt{\phi^2 + \epsilon^2}} \tag{5.11}$$

can be used. Figure 5.5 shows that this function actually never reaches zero and one, but

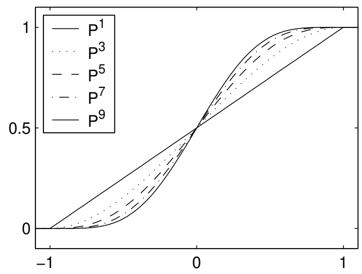


Figure 5.4: Smooth Heaviside function, $\epsilon = 1$, for different polynomial degrees

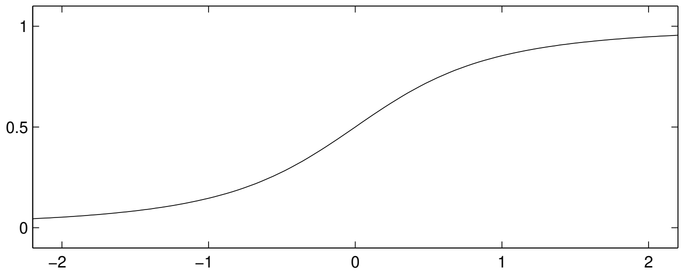


Figure 5.5: Very smooth Heaviside function, $\epsilon = 1$

only approaches asymptotically. It is, however, infinitely smooth. The δ-function can be defined as:

$$\delta(\phi) = \frac{\partial}{\partial \phi} H(\phi). \tag{5.12}$$

Hence smoothed versions of the δ-function can be defined from the Heaviside function. Figure 5.6 shows similar plot for the smoothed δ-functions, where the δ-function denoted

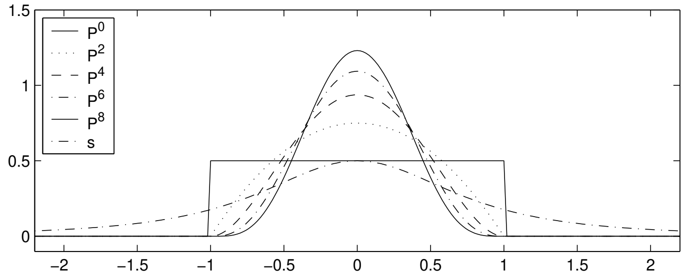


Figure 5.6: Smooth delta functions, $\epsilon = 1$

P^i is based on a polynomial of degree i, and s denotes the very smooth version.

Note that all the smoothed δ-functions have the property:

$$\int_{-\infty}^{\infty} \delta(\phi, \epsilon)\ \mathrm{d}x = 1. \tag{5.13}$$

Most of them actually have the property that

$$\int_{-\epsilon}^{\epsilon} \delta(\phi, \epsilon)\ \mathrm{d}x = 1, \tag{5.14}$$

since most of the smoothed δ-function are zero outside $\pm\epsilon$.

End of Example 5.1.

The smoothed δ-function must give the same contribution as the exact δ-function, for example when used for applying the surface tension force, i.e.

$$\int_{\Gamma} \kappa_{\Gamma}\ \mathbf{n}_{\Gamma}\ \mathrm{dS} = \int_{\Omega} \kappa_{\Gamma}\ \delta(\phi, \epsilon)\ \mathbf{n}_{\Gamma}.\ \mathrm{d}\mathbf{x}. \tag{5.15}$$

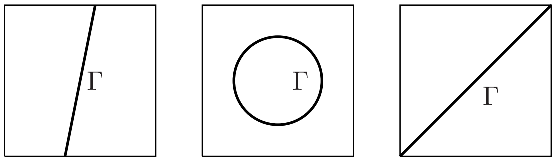
We will therefore require that our smoothed δ-function fulfill

$$l(\Gamma) = \int_{\Gamma} 1\ \mathrm{dS} = \int_{\Omega} \delta(\phi, \epsilon)\ \mathrm{d}\mathbf{x}, \tag{5.16}$$

where $l(\Gamma)$ is the length of the interface.

Example 5.2: Error in smoothed δ-functions

Consider the three different interfaces in Figure 5.7. We will test the accuracy of integra-



(a) (b) (c)

Figure 5.7: Different interfaces

tion of $\delta(\phi, \epsilon)$ in Equation (5.16) for different values of ϵ and the degree of the polynomial used when creating the δ-function, P^i, from Figure 5.6. The test is carried out in a $[-\frac{1}{2}; \frac{1}{2}] \times [-\frac{1}{2}; \frac{1}{2}]$ domain. The circle has radius 0.3.

$\epsilon \setminus \mathsf{P}^i$	0	2	4	6	8
0.1	1.7e-03	8.4e-04	1.5e-04	5.8e-05	1.7e-04
0.2	1.9e-03	9.2e-05	3.1e-06	6.2e-07	3.3e-06
0.3	4.1e-03	4.7e-06	5.0e-06	3.3e-07	2.9e-07
0.4	3.7e-03	2.1e-05	4.4e-06	1.1e-07	8.7e-08

Table 5.1: Error in smoothed δ-function for linear interface, Figure 5.7(a)

$\epsilon \setminus \mathsf{P}^i$	0	2	4	6	8
0.05	1.4e-02	3.0e-03	2.5e-03	2.5e-04	1.4e-03
0.10	1.6e-03	5.8e-04	2.6e-04	5.2e-05	6.4e-04
0.15	5.1e-03	4.8e-05	7.1e-05	5.2e-05	5.0e-05
0.20	8.5e-03	1.1e-04	2.1e-05	4.5e-06	5.0e-06

Table 5.2: Error in smoothed δ-function for circular interface Figure 5.7(b)

$\epsilon \setminus \mathsf{P}^i$	0	2	4	6	8
0.1	7.6e-02	7.5e-02	6.3e-02	5.5e-02	4.9e-02
0.2	2.0e-01	1.5e-01	1.3e-01	1.1e-01	9.8e-02
0.3	3.0e-01	2.3e-01	1.9e-01	1.6e-01	1.5e-01
0.4	4.0e-01	3.0e-01	2.5e-01	2.2e-01	2.0e-01

Table 5.3: Error in smoothed δ-function for linear interface Figure 5.7(c)

Results in Table 5.1, 5.2, and 5.1 show the error when integrating the $\delta(\phi, \epsilon)$-function in Equation (5.16), using a spatial discretization having 226 elements of polynomial order 5. Table 5.1 and 5.2 shows that the error decay for increasing polynomial degree P^i, and to some degree also with the thickness of the interface ϵ. However, in Table 5.3 the error is smallest for smallest ϵ, more or less independent of the polynomial degree P^i. This is

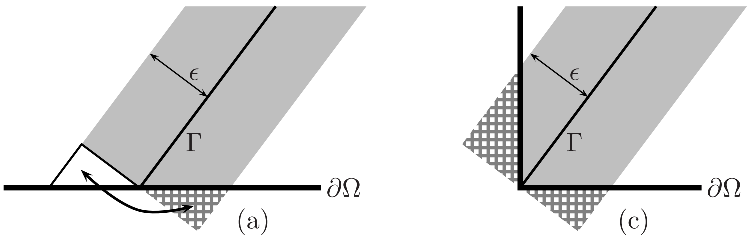


Figure 5.8: Integrating close to the boundary

due to the fact that integrating up to the boundary will be inaccurate. Figure 5.8 shows the problem, where the gray part is where the smoothed δ-function should be integrated, however the gray-white part outside the domain are integrated. In case (a) the white boxed area is integrated and cancels out the gray-white part outside the domain, while in case (c) it does not. The error in case (c) will be smaller for smaller ϵ, since the part outside the domain decreases with ϵ.

The comforting news is to see that for the circular interface, case (b), the smoothed δ-function works well.

End of Example 5.2.

It is important for the results in Example 5.2 that the level set function ϕ is a true distance function. If ϕ is not a distance function, the interface thickness will change and the property (5.13) will not be true: Assume ϕ is a distance function, then $\psi(x) = a\phi(x)$, $a \neq 1$, is not a distance function, $\nabla\psi = a\nabla\phi = a$. Figure 5.9

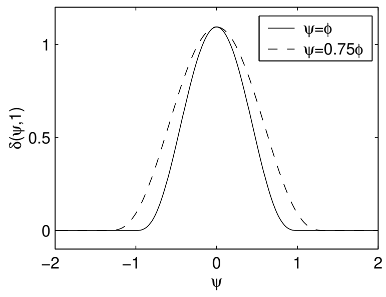


Figure 5.9: δ-function when ψ is not a distance function

shows the smoothed δ-function for $a = 1$ and $a = \frac{3}{4}$, and the area under the smoothed δ-function is bigger for $a = \frac{3}{4}$ than for $a = 1$. This means that one of the consequences of the level set not being a distance function is that we do not apply the correct surface tension force.

5.1.2 Level Set Band

When solving the flow equations, terms involving the interface are based on the level set function: Interface normal and surface tension is calculated at the interface $\phi = 0$, density and viscosity are based on the Heaviside, and the surface tension force is applied using a δ-function . All in all, the level set function is only used where $\phi = 0$. We are using the smoothed versions of the Heaviside and δ-function, hence values of the level set function is used in the distance ϵ from the interface. However, for $|\phi| > \epsilon$, only the sign of the level set function is used, at least when not using the fully smooth versions of the Heaviside and δ-functions.

Therefore the level set need only to be accurate within $|\phi| < \epsilon$. It actually only need to be a distance function in this region, elsewhere it is not important whether it is a distance function, and whether it is close to being so or far from it. We will therefore make sure that the level set is treated accurately within $|\phi| < \epsilon$.

In order to update the level set numerically in Eq. (XXX) with good precision within the distance ϵ from the interface, the level set need to be accurate in a distance slightly larger, say 1.5ϵ. Otherwise errors in distance only slightly larger than ϵ may influence on the solution within the distance ϵ. Hence we shall require the level set to be a signed distance function within 1.5ϵ of the interface, and elsewhere we shall only need the sign. From the point of view of the discretized domain, we need only consider the level set in a band of elements around the interface: We can limit our attention to elements within 1.5ϵ of the interface, e.g. the gray elements in Fig. 5.10. Outside this band of elements, we shall only need the sign. We shall take advantage of this where possible in order to lower the computational work of handling the level set.

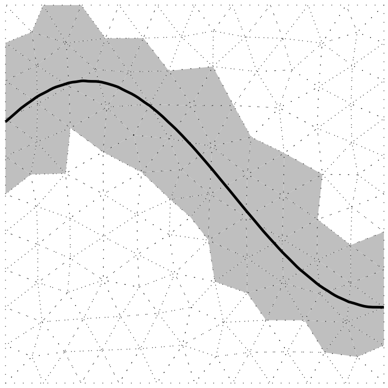


Figure 5.10: Elements within 1.5ϵ of the interface

5.2 Reinitialization of the Level Set

A fluid particle at a certain distance from the interface will, as time goes, seldom remain at that distance. The level set ϕ follows the flow, hence the value of ϕ at the fluid particle will follow this particle, and will end up at another distance from the interface. Hence even if the level set ϕ was a signed distance function at some point, it will not remain so.

Example 5.3: Level Set in Canal Flow

Consider a flow in a canal as in the top plot of Figure 5.11. In the flow we put a droplet, i.e. a circular interface, as shown in the bottom-left part of the figure, where the droplet and 3 level set isocontours for $\phi = \{-0.1, 0.0, 0.1\}$ are plotted. After a period of time, updating the level set using the level set equation (2.15), the droplet will be deformed to an arrow like shape as in the bottom middle plot at $t = 1$ and right plot at $t = 3$. The deformation to the arrow like shape is correct and not a problem. However, notice the isocontours: They are at $t = 0$ (left) equidistant to the interface, meaning the level set is a signed distance function. At $t = 1$ and $t = 3$ we see that the isocontours no longer are equidistant to the interface, the distance from the zero contour line to the other two lines varies between 0.05 and 0.3 at $t = 1$ and between 0.02 and 0.9 at $t = 3$. In other words, the level set is no longer a distance function.

End of Example 5.3.

To retain the level set as a signed distance function, it must be reinitialized. If the level set is not reinitialized, areas of small and large gradients will form, changing the thickness of the interface and eventually degrading the accuracy and stability of the method.

The goal of the reinitialization procedure is to produce a true distance function to the interface. The procedure must not move the interface, hence the zero contour must remain unchanged, and ϕ must be found such that $|\nabla\phi| = 1$. In [64] the

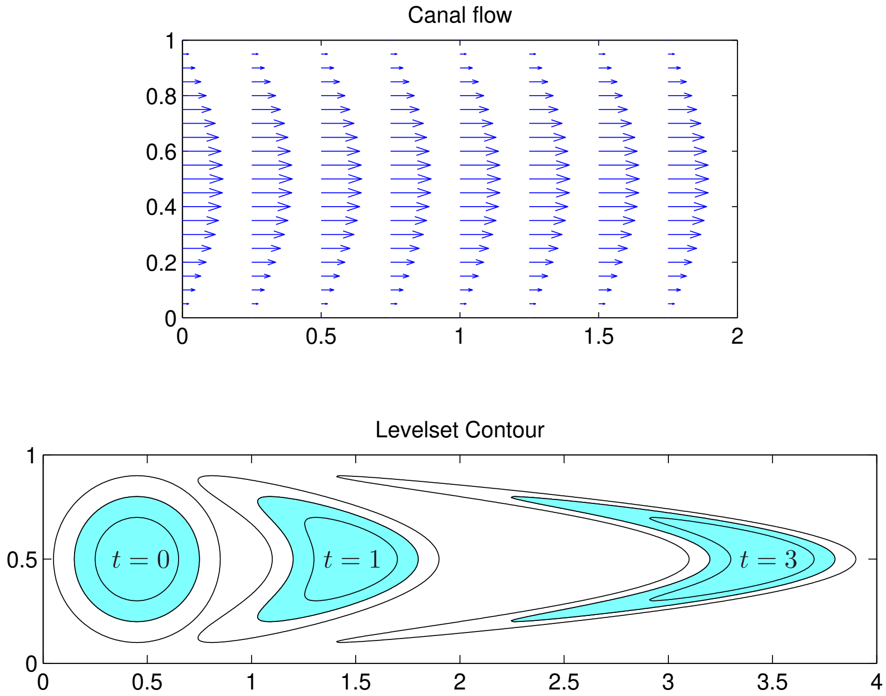


Figure 5.11: Level Set in canal flow

following reinitializing equation was proposed

$$\phi_\tau + \mathrm{sign}(\phi_0)(|\nabla\phi| - 1) = 0, \tag{5.17}$$

where ϕ_0 is the initial level set to be reinitialized and τ is an artificial time. Evolving this equation to steady state will produce a signed distance function ϕ. The reinitializing equation propagate information along the characteristic lines, which are normals of the interface. The characteristic speed is 1, hence the level set will be reinitialized from the interface and along the interface normals.

As we seek the level set reinitialized in a distance 1.5ϵ from the interface, it is sufficient to run the reinitialization to time $\tau = 1.5\epsilon$. However, if the initial level set is already close to a signed distance function, fewer iterations are often sufficient.

The reinitialization equation (5.17) is a Hamilton-Jacobi equation, i.e. it has the form

$$\phi_\tau + G(\phi_x, \phi_y) = 0. \tag{5.18}$$

Solutions to Hamilton-Jacobi equations are continuous, but may have discontinuous derivatives. The solution in our case is a signed distance function, and Figure 5.12 is a 1D example showing how two surface points results in a level set function with

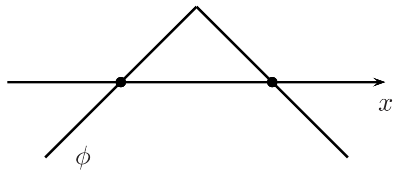


Figure 5.12: Two surface points produces a level set function with a sharp peak

a peak, continuous everywhere, but with a discontinuous derivative where $\mathbf{x}_\Gamma^{\min}(\mathbf{x})$ is not unique.

A standard high-order method does not handle such discontinuities easily, especially when differentiating as in Eq.(5.17), and the method may turn inaccurate, difficult to solve, or even unstable. Discontinuous Galerkin methods has been applied to Hamilton-Jacobi equations [39], the procedure involves the solution to a least square problem. Also the use of WENO methods has been proposed [77] to overcome such problems but remains a challenge for general unstructured grids.

We will here try to use a more direct approach. Rewrite the equation in the LDG way,

$$\frac{\partial \phi}{\partial t} = \mathrm{sign}(\phi_0, \epsilon)(1 - |\mathbf{h}|) \ , \tag{5.19a}$$

$$\nabla \phi = \mathbf{h} \ , \tag{5.19b}$$

and discretize this as

$$\left(\frac{\partial \phi}{\partial t}, L_j\right)_{\mathsf{D}^k} = (\mathrm{sign}(\phi_0, \epsilon)(1 - |\mathbf{h}|), L_j)_{\mathsf{D}^k} \tag{5.20a}$$

$$(\nabla \phi, L_j)_{\mathsf{D}^k} = (\mathbf{h}, L_j)_{\mathsf{D}^k} + (\mathbf{n}(\phi - \phi^*), L_j)_{\partial \mathsf{D}^k} \ , \tag{5.20b}$$

or in discrete operators

$$\frac{\partial \phi}{\partial t} = \mathbf{s}(1 - |\mathbf{h}|) \tag{5.21a}$$

$$\mathsf{S}^k \phi = \mathsf{M}^k \mathbf{h} + \mathsf{F}^k(\mathbf{n}(\phi - \phi^*)), \tag{5.21b}$$

where $\mathbf{s} = sign(\phi_0, \epsilon)$. The vector $\mathbf{h}$ can again be eliminated locally. Due to the lack of smoothness in this nonlinear system, we would like to apply a Lax-Friedrich-like flux to add some dissipation in order to control stability. Adding a dissipative term only to Equation (5.21b) as the Lax-Friedrich procedure describes, is not enough. We would like to apply the dissipative term in a very similar manner as for the conservation law Equation (3.94), on the "outermost" Equation (5.21a), i.e. on the form

$$\frac{\partial \phi}{\partial t} = \mathbf{s}(1 - |\mathbf{h}|) + \mathsf{F}^k(\frac{c}{2}(\phi^+ - \phi^-)) \ , \tag{5.22a}$$

$$\mathsf{S}^k \phi = \mathsf{M}^k \mathbf{h} + \mathsf{F}^k(\mathbf{n}(\phi - \frac{1}{2}(\phi^- + \phi^+))), \tag{5.22b}$$

i.e. applying the central flux part to Equation (5.22b) and the diffusive part to Equation (5.22a). We will determine the wave-speed c in a similar manner as for the conservation law, i.e.,

$$c = \max\left(\left|\frac{\partial G(\phi_x,\phi_y)}{\partial \phi_x}\right|, \left|\frac{\partial G(\phi_x,\phi_y)}{\partial \phi_x}\right|\right). \tag{5.23}$$

We have $G(\phi_x,\phi_y) = \text{sign}\left(\sqrt{\phi_x^2+\phi_y^2}-1\right)$, giving

$$\left|\frac{\partial G(\phi_x,\phi_y)}{\partial \phi_x}\right| = \frac{2|\phi_x|}{2\sqrt{\phi_x^2+\phi_y^2}}, \qquad \left|\frac{\partial G(\phi_x,\phi_y)}{\partial \phi_y}\right| = \frac{2|\phi_y|}{2\sqrt{\phi_x^2+\phi_y^2}}, \tag{5.24}$$

and since now $\nabla\phi = 1$, then $|\phi_x|, |\phi_y| \leq 1$ and then also $|\frac{\partial G(\phi_x,\phi_y)}{\partial \phi_y}| \leq 1$ giving $c \leq 1$. We will therefore choose $c = 1$. We will not give any proof that this is actually a consistent way of solving the system, however numerical tests later will show that it works quite well.

5.2.1 Filtering the Reinitialization Process

The reinitialization is non-linear and the solution is not smooth everywhere. It is therefore necessary to use a filtering technique for further stabilizing the reinitialization process and retain high order accuracy. We basically use a low-pass exponential filter, the precise description can be found in [23].

Using an explicit method to solve (5.17) and a exponential filter damping high modes, $\mathsf{T} = \mathsf{I} - \mathsf{H}$, I being the identity operator, corresponds to

$$\tilde{\phi}_{i+1} = \phi_i - \Delta t \ \text{sign}(\phi_0)(|\nabla\phi_i| - 1) \tag{5.25a}$$

$$\phi_{i+1} = \big(\mathsf{I} - \mathsf{H}\big)\tilde{\phi}_{i+1} \tag{5.25b}$$

Iterating the original reinitialization equation (5.17) to steady state implies eventually $|\nabla\phi| - 1 = 0$. When adding the filter, the steady state solution will emerge when the filter and the forcing balances, $\Delta t \ \text{sign}(\phi_0)(|\nabla\phi_i| - 1) \approx \mathsf{H}\phi_i$. The errors due to filtering becomes a part of the solution. Figure 5.13 shows an example of what happens when filtering errors become dominant: We have reinitialized the level set for a circular interface, starting with $\phi = 0$. The correct reinitialized solution is shown left, and right is the result when solving the reinitialization equation above on a low resolution grid and using a quite strong filter . What happens is that the filter removes high modes from the solution, leaving basically only the zeroth (constant) and first (linear) mode of the solution, the result being a kind of staircase effect and large discontinuities between the elements. The filter here is somewhat stronger than necessary in order to emphasize the problem visually, however even when using a filter strong enough to just stabilize the system, the effect will still be there, destroying accuracy.

Filtering of steady state problems must be designed with care. We have designed the reinitialization method such that filtering only takes place when necessary,

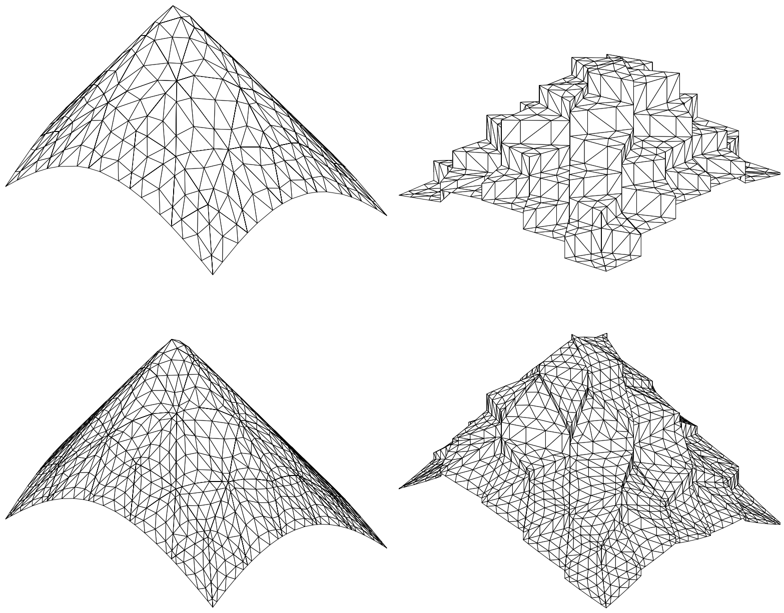


Figure 5.13: Standard filtering of the reinitialization of a circle - 62 elements of order 4 (top) and 6 (bottom), filter order 5.

and then only as little as possible, and preferably never in elements containing the interface. This is applied adaptively, by calculating a measure of the "need for filtering" and using this measure to determine the degree of filtering in each element. Since now G in Equation (5.18) is zero in the steady state case, i.e. when $|\nabla\phi| = 1$, we can use G as this measure. Let $H(p, r)$ be a filter of order p, leaving out the first r modes, the procedure is:

foreach element k
  if $|G^k| < l_0$
    do not filter on element k at all
  elseif $|G^k| < l_1$
    use $H = H(15, 3)$ on element k
  elseif $|G^k| < l_2$
    use $H = H(15, 2)$ on element k
  elseif $|G^k| < l_3$
    use $H = H(15, 1)$ on element k
  elseif $|G^k| < l_4$
    use $H = H(15, 0)$ on element k
  else $|G^k| < l_5$
    use $H = H(6, 0)$ on element k

where the parameters l_j decides when the different filters are applied. The filter $H = H(15, 0)$ almost equals the $p = 16$ case in the left plot in Figure 3.11. The parameters l_j will depend on the time step, and we will also choose l_j, depending on how close element k is to the interface. Especially will we use larger l_j when element k contains the interface, in order to filter less in those elements. A typical choice

$$\mathbf{l} = (0.05; 0.1; 0.2; 0.4; 2.5) \tag{5.26}$$
$$\mathbf{l}_0 = (0.2; 0.4; 0.8; 1.5; 2.5) \tag{5.27}$$

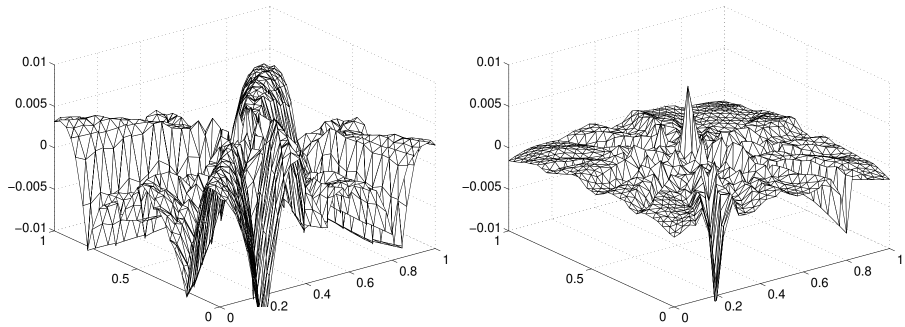


Figure 5.14: Error of reinitialization of circle using normal filtering (left) and adaptive filtering (right). Spatial Discretization using 226 elements of order 3

Figure 5.14 shows the error of the reinitialization process when reinitializing the circular interface also depicted in Figure 5.2.

Time consistent filtering

Other approaches for filtering of steady state problems have recently been proposed, [41]. They have developed time-consistent filters especially designed for time-integrating solutions of DG methods to steady state. The idea is that the total filter after time T should not depend on the time steps taken, i.e. the filter should in total have same strength independent of whether we take k_1 steps of size Δt_1 or k_2 steps of size Δt_2, $k_1 \Delta t_1 = k_2 \Delta t_2 = T$. Hence the filter strength for one time step depends on the time step size Δt. Such filters would be interesting to test on the reinitialization process.

5.2.2 Streamline Diffusion - Diffusion Along Characteristics

As mentioned before, the characteristic lines are normal to the surface. Furthermore, the gradient along a characteristic line is 1: The direction of steepest descent/ascent is the direction leading to the closest point on the interface. Defining

$\mathbf{w} = \frac{\nabla\phi}{|\nabla\phi|}$, above means

$$\frac{\partial\phi}{\partial\mathbf{w}} = \mathbf{w}\cdot\nabla\phi = 1. \tag{5.28}$$

This is true almost everywhere, except where $\mathbf{x}_\Gamma^{\min}(\mathbf{x})$ is not unique. If we differentiate once more along the characteristic, we get

$$\frac{\partial^2\phi}{\partial\mathbf{w}^2} = \mathbf{w}\cdot\nabla(\mathbf{w}\cdot\nabla\phi) = 0. \tag{5.29}$$

Again this is true almost everywhere, and it is especially true on the interface $\phi = 0$. Adding a term like this to the reinitialization equation will therefore have no effect almost everywhere, again especially on the interface $\phi = 0$, hence it will not move the interface. The reinitialization equation will then become:

$$\phi_\tau = \mathrm{sign}(\phi_0)(1 - |\nabla\phi|) + \nu_w\frac{\partial^2\psi}{\partial\mathbf{w}^2}\ , \tag{5.30}$$

where ν_w is some diffusion constant. This implies that where $\mathbf{x}_\Gamma^{\min}(\mathbf{x})$ is not unique, i.e. the level set function has some kind of peak, the diffusion term will smoothen the peak and make the level set function infinitely smooth. Far away from the peaks, the diffusion term will have no effect. The diffusion constant ν_w should be determined in order to have the desired effect, but on an region as small as possible. Figure 5.15 shows the error of the reinitialization problem also depicted

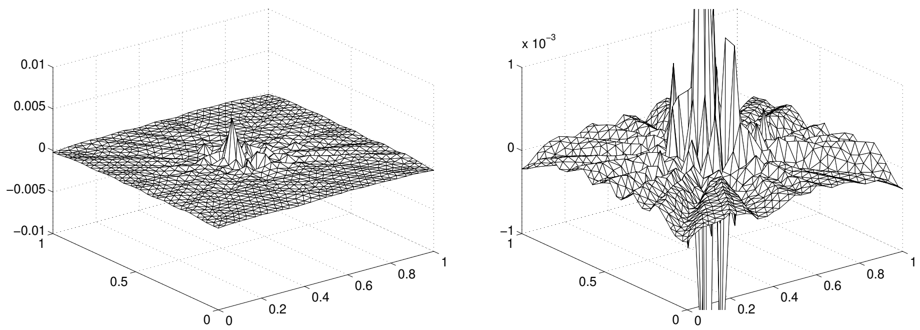


Figure 5.15: Error of reinitialization of circle using diffusion along characteristics $\nu_w = 0.0025$ and adaptive filtering. Same setup as Figure 5.14. Left has same axes as Figure 5.14, right is a zoom

in Figure 5.14, but with the added diffusion along characteristic-term. The term effectively implies that less filtering is applied in the remaining part of the domain, thereby further reducing errors coming from the filtering. As seen in Figure 5.15, the error is fairly big in the center, but small elsewhere, hence the diffusion has the desired effect.

We have not further investigated into the optimal size of the diffusion constant ν_w. The optimal ν_w will depend on the spatial discretization, the size of the elements and the polynomial order in the element. In general, the more resolution the smaller diffusion coefficient. Choosing ν_w too small, it will have no effect, and

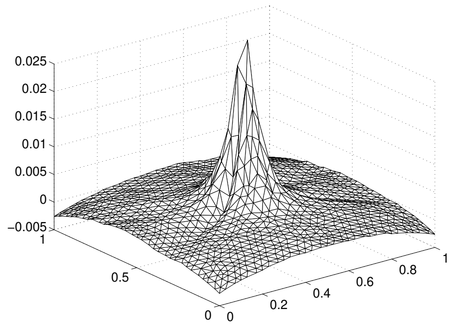


Figure 5.16: Error of reinitialization of circle using diffusion along characteristics $\nu_w = 0.05$ and adaptive filtering. Same setup as Figure 5.14.

too big the resulting reinitialized level set will look like in Figure 5.16, the diffusion will influence too large a region.

5.2.3 Restricting Level Set Function Values

As mentioned in Section 5.1.2, we will only require the level set function to be a distance function accurately within a region slightly larger than ϵ, for example $\omega = 1.5\epsilon$. Outside ω, the value of the level set function is not important, hence we will restrict the level set function to an maximum absolute value $\phi_{\max}$, as depicted

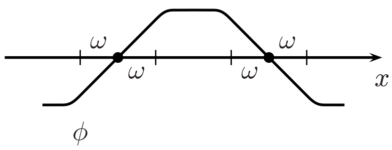


Figure 5.17: Peak is removed

in Figure 5.17. The figure is equivalent to Figure 5.12, having the same interface points. The reason is twofold. Firstly, peaks far away from the interface is removed, thereby adding more smoothness to the level set function and hence lowering the need for filtering.

Secondly it simplifies the handling of a level set band. Assume we have chosen a band of elements around the interface, as in Figure 5.10. When the interface moves, some elements close to the band will eventually be within the specified distance of the interface to become a part of the band, hence we must be able to add elements to the band and delete elements from the band. But what value should the level set have in a newly added element? Choosing the band to include as well the ω-part as the transition to the flat part, the level set in the new element should have the value $\pm\phi_{\max}$. For example, assume the level set function in Figure 5.17 is the band we consider, adding an element to the left end would result in a new level set band

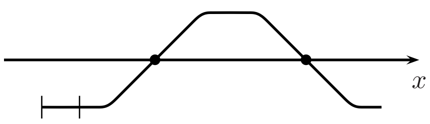


Figure 5.18: Element added to the left

as depicted in Figure 5.18.

Restricting the level set.

If we have a distance function as in Figure 5.12, it can be restricted as follows

$$\phi_r = \begin{cases} \phi & |\phi| \le \omega \\ p(\phi) & \omega < |\phi| < 1.5\omega \\ \operatorname{sign}(\phi)\, 1.25\,\omega & |\phi| > 1.5\omega \end{cases} \tag{5.31}$$

where $p(\phi)$ is some (polynomial) function assuring a smooth transition for $\omega < |\phi| < 1.5\omega$, and $\phi_{\max} = 1.25\,\omega$ is used. The function p can be found in the same manner as in Example 5.1.

Maintaining the restriction.

The level set equation (2.15) does not have any influence on the restriction. However, when we reinitialize, the level set will readapt towards a signed distance function everywhere.

The first approach is to reinitialize until the level set function is a distance function within $|\phi| < 1.5\omega$, and then use the procedure in Equation (5.31). However, if we wish to reinitialize less, then subsequent restrictions of the same level set may imply that $\max|\phi| < 1.25\,\omega$. For example will the application of the restriction procedure twice on the level set imply that $\max|\phi| \approx 1.2\,\omega$. This gives problems when we wish to add new elements to the band and assign them the value $1.25\,\omega$, thereby introducing a discontinuity.

Another approach is to solve a reinitialization of the form

$$\phi_\tau = \operatorname{sign}(\phi_0)(l(\psi,\omega) - |\nabla\phi|), \tag{5.32}$$

where

$$l(\phi,\omega) = \begin{cases} 1, & |\phi| < \omega, \\ 0, & |\phi| > \frac{5}{4}\omega, \\ 1 - 4\frac{|\phi|-\omega}{\omega}, & \text{in between}, \end{cases} \tag{5.33}$$

The l-function in Equation (5.33) is here linear between 1 and 0, however we will generally use smoother versions as the P^3 depicted in Figure 5.19. This new

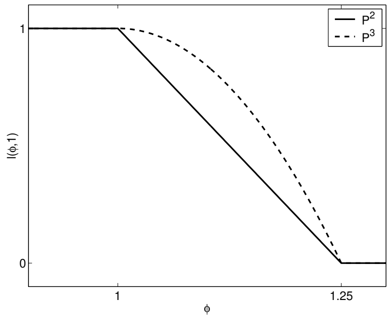


Figure 5.19: The l-function

formulation will force the level set function to have $\nabla\phi = 1$ within ω, and $\nabla\phi = 0$ when $|\phi| > \frac{4}{5}\omega$, effectively flattening the level set for $|\phi| > \omega$. Figure 5.20 shows

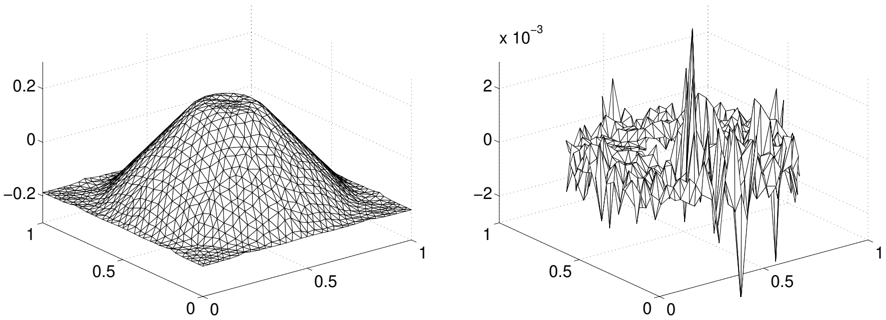


Figure 5.20: The restricted level set (left) corresponding to the interface in Figure 5.2, and error within $\omega = 0.15$ (right).

the level set corresponding to the interface in Figure 5.2 when reinitialized using the restricting technique, and also the error of the reinitialization within ω.

5.2.4 Varying Interface Thickness, ϵ.

As mentioned in Section 5.1.1, we may let ϵ vary in the domain, such that ϵ follows the size of the element. Consider e.g.

$$\epsilon = \epsilon_0(1 + \frac{4}{5}x - \frac{2}{5}y), \qquad \omega = 1.5\,\epsilon \tag{5.34}$$

where ϵ will vary between $\frac{3}{5}\epsilon_0$ and $\frac{9}{5}\epsilon_0$ for $(x, y) \in [0; 1] \times [0; 1]$. To handle the varying ϵ in the reinitialization, we will add one more term

$$\phi_\tau = \text{sign}(\phi_0, \epsilon)(l(\psi, \omega) - |\nabla\phi|) + \nu_w \frac{\partial^2\psi}{\partial \mathbf{w}^2} + \lambda(1 - l(\phi, \omega))\left(\frac{5}{4}\omega\ \text{sign}(\phi) - \phi\right). \tag{5.35}$$

This new term simply forces the level set outside $|\phi| > \frac{5}{4}\epsilon$ toward the value $\frac{5}{4}\epsilon$, to make sure the level set value far away from the interface is close to $\frac{5}{4}\epsilon$. In this way we avoid discontinuities when adding new elements to the band, and if setting Dirichlet conditions on far away boundaries. The value of λ has been found experimentally in the range of $5 - 200$ to have the desired effect. We will in general use the value $\lambda = 50$. For constant ϵ it may be omitted. Figure 5.21 shows the

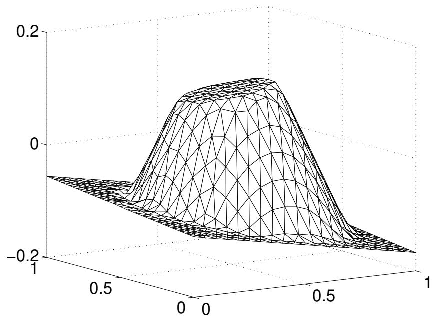


Figure 5.21: Reinitialization of circle with varying interface thickness ϵ as in Equation (5.34)

level set corresponding to the interface in Figure 5.2 after being reinitialized using a varying interface thickness ϵ.

The interface thickness ϵ must vary smoothly throughout the domain in order to work.

5.2.5 Boundary Conditions for the Reinitialization

Boundary conditions for the original reinitialization equation (5.17) is needed when the interface intersects the domain boundary $\partial\Omega$. The top dotted part of the

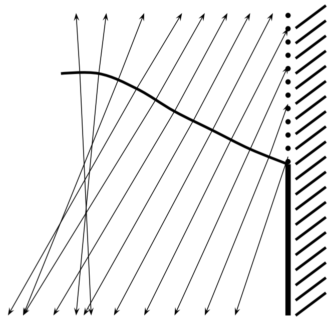


Figure 5.22: Characteristic lines of the level set function

right boundary in Figure 5.22 intersects characteristics of the level set, while the bottom part does not, i.e., boundary conditions are needed for inflow boundaries corresponding to the bottom part, and no boundary conditions should be applied at outflow boundaries corresponding to the upper part.

The boundary conditions can be found by reinitializing at the boundary first using

$$\left.\frac{\partial \phi}{\partial \tau}\right|_{\mathbf{x}\in\partial\Omega} = \mathrm{sign}(\phi_0, \epsilon)(\beta\ l(\phi,\omega) - |\nabla\phi|)|_{\mathbf{x}\in\partial\Omega} \tag{5.36}$$

where β is the slope of the level set on the boundary,

$$\beta = |\nabla\phi - \mathbf{n}(\mathbf{n}\cdot\nabla\phi)| \,. \tag{5.37}$$

and $\mathbf{n}$ is the outward pointing normal vector. The calculation of β is illustrated in Figure 5.23. Before taking a time step of the global reinitialization, we take

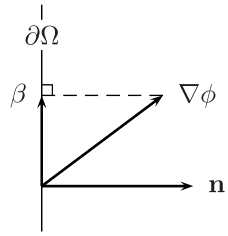


Figure 5.23: Calculation of β

one time step of the reinitialization of the boundaries, the result on the boundaries being used as Dirichlet boundary data for the global reinitialization.

When restricting attention to a band of elements within a certain distance of the interface, also boundary conditions on the outermost elements of the band must be given, i.e. boundary conditions on the interface between white and gray elements of Figure 5.10. Using the modified reinitialization equation, the value of the level set far away from the interface is $\frac{5}{4}\epsilon$, and this value may be used as boundary value on the outermost element boundaries.

5.2.6 Discretization of the Reinitialization Equation

The discretization of Equation (5.35) follows the previous discussion. Rewrite the equation such that

$$\frac{\partial \phi}{\partial t} = \text{sign}(\phi_0, \epsilon)(l(\phi, \epsilon) - |\mathbf{h}|) + \nu_w \mathbf{w} \cdot \mathbf{g} + f \ , \tag{5.38a}$$

$$\nabla \phi = \mathbf{h} \ , \tag{5.38b}$$

$$\nabla(\mathbf{w} \cdot \mathbf{h}) = \mathbf{g} \ , \tag{5.38c}$$

We discretize this as

$$\left(\frac{\partial \phi}{\partial t}, L_j\right)_{\mathsf{D}^k} = (\text{sign}(\phi_0, \epsilon)(l - |\mathbf{h}|) + \nu_w \mathbf{w} \cdot \mathbf{g} + f, L_j)_{\mathsf{D}^k} \ , \tag{5.39a}$$

$$(\nabla \phi, L_j)_{\mathsf{D}^k} = (\mathbf{h}, L_j)_{\mathsf{D}^k} + (\mathbf{n}(\phi - \phi^*), L_j)_{\partial \mathsf{D}^k} \ , \tag{5.39b}$$

$$(\nabla(\mathbf{w} \cdot \mathbf{h}), L_j)_{\mathsf{D}^k} = (\mathbf{g}, L_j)_{\mathsf{D}^k} + (\mathbf{n} \cdot \mathbf{w}(\mathbf{h} - \mathbf{h}^*), L_j)_{\partial \mathsf{D}^k} \tag{5.39c}$$

or in discrete operators

$$\frac{\partial \phi}{\partial t} = \mathbf{s}(\mathbf{l} - |\mathbf{h}|) + \nu_w \mathbf{w} \cdot \mathbf{g} + \mathbf{f} \ , \tag{5.40a}$$

$$\mathsf{S}^k \phi = \mathsf{M}^k \mathbf{h} + \mathsf{F}^k(\mathbf{n}(\phi - \phi^*)), \tag{5.40b}$$

$$\mathsf{S}^k(\mathbf{w} \cdot \mathbf{h}) = \mathbf{M}^k \mathbf{g} + \mathsf{F}^k(\mathbf{n} \cdot \mathbf{w}(\mathbf{h} - \mathbf{h}^*)) \tag{5.40c}$$

where $\mathbf{s} = \text{sign}(\phi_0, \epsilon)$. If global boundary conditions are needed, a $\mathsf{F}^k_{\partial\Omega}(\phi - \phi^*_{\partial\Omega})$ term can be included in (5.40a).

5.2.7 Time-stepping the Reinitialization

For stepping the reinitialization proces forward in time, we shall use the fourth-order low storage Runge-Kutta scheme from [10]. The scheme is developed to integrate equations of the form $\frac{\partial}{\partial t}u + \frac{\partial}{\partial x}u = 0$. The characteristics of this type of equations are straight lines, along which the solution travel at constant speed. This is in many aspects also what happens in the level set reinitialization, making the scheme appropriate.

5.2.8 Reinitialization Test

Figure 5.24 shows the upper right part of the reinitialization of a circular interface in a 1×1 domain, centered at $(0.5; 0.5)$ and with a radius 0.3. The domain is discretized using 894 elements having order 3. We are using a large ω such that

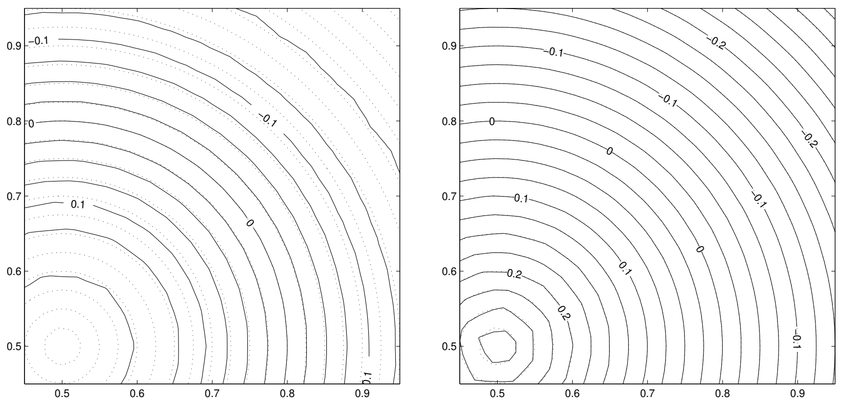


Figure 5.24: Reinitialization of circle after time $\tau = 0.155$ (left) and $\tau = 0.5$ (right). Dotted lines are exact distance contours, solid lines are level set contours.

the level set will be a distance function in the whole domain. On the right plot, the true and exact solution is indistinguishable apart from close to the center of the circle at $(0.5; 0.5)$ due to the local loss of smoothness. Oscillations coming from the discontinuity is controlled by the filtering and the streamline diffusion term. The price for stability being some loss of accuracy, but only close to the discontinuity. The left plot shows the level set contours at $\tau = 0.155$, and illustrates that the level set is first reinitialized at the interface and the reinitialization progresses along the characteristic lines, almost with speed one. The speed is not exactly one due to the smoothed sign function. At this point of $\tau = 0.155$, the level set would be accurate in a band wide enough for further use: The element side lengths are of the order 0.05, and it need only to be accurate in a distance of order 0.075 from the interface.

5.3 Preliminary Test of Level Set Modeling

As a preliminary test of the level set we will apply the level set equation (2.15) using constant velocity $\mathbf{u} = (1, 1)$ to two different interfaces in a periodic domain. The domain is of size $[-0.5; 0.5] \times [-0.5; 0.5]$, hence in time $t = 1$ the interface should be at the initial position. Figure 5.25 depicts the domain and two different interfaces. Figure 5.26 shows the initial circular interface (dashed) and the interface at time $t = 1$ (solid) for two different discretizations. In [52], the figures in Appendix C is presented. They show a similar test for a Volume of Fluid (VOF) method on a 200×100 grid, where the circular interface occupies approximately 63 cells. The VOF methods all show some kind of interface deformation. Comparing the number of grid-points inside the interface used here and in [52]: The left plot in Figure 5.26 uses 37 nodes inside and there is a slight deformation in the direction of the flow, while the right plot uses 81 nodes inside, and the initial and final interface are indistinguishable. The VOF methods in [52] use 63 cells and has each different

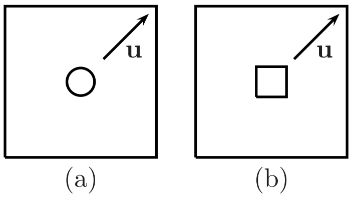


Figure 5.25: Domains for constant convection test

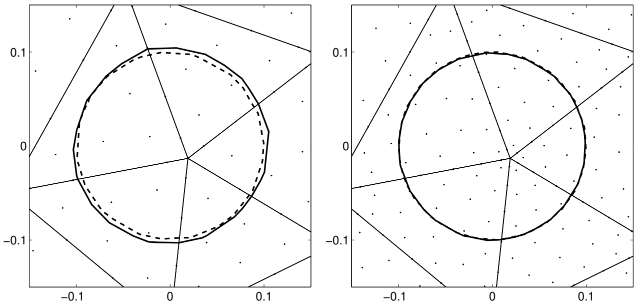


Figure 5.26: Convection of circular interface 62 using elements of order 5 (37 nodes inside)left and order 8 (81 nodes inside), right

problems keeping the shape.

This shows one of the strengths of the level set method combined with a high order spatial scheme: We have not applied any special tricks or methods. As long as the interface is smooth, it is very well approximated.

Figure 5.27 shows similar results for the square interface. Since the square is not smooth in the corners, the high order spatial scheme has some difficulties representing the interface at these corners, and higher resolution is needed. Not

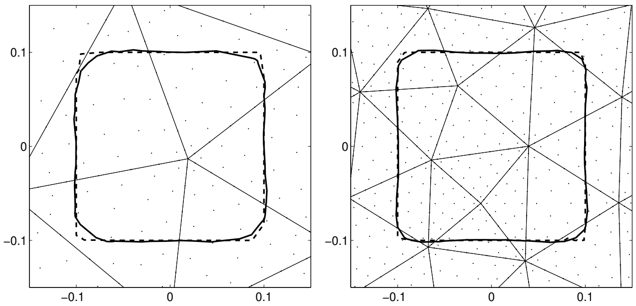


Figure 5.27: Convection of square interface using 62 elements of order 8 (92 nodes inside) left and 226 elemenst of order 8 (405 nodes inside) right

surprisingly are the corners rounded, while the linear parts are approximated well.

Comparing with the VOF examples in [59], having 400 cells inside the square, the results here are quite good. The methods in [59] and [52] all have some kind of grid-dependent deformation, which we do not have, since we are using a fully unstructured triangular mesh.

The preliminary conclusion is that the level set and the level set equation (2.15) work well with the high order spatial scheme.

5.4 Final Remarks on the Level Set Modeling

The level set equation (2.15) is easy to handle and use. The tricky part of the level set modeling is the reinitialization process. There exists other approaches for the reinitialization:

- It is possible to apply ENO[1]-schemes in order to minimize effects of oscillations and Runge phenomena at discontinuities. Examples are [63, 66]. ENO and WENO schemes are fairly easily applied on structured grids. On general unstructured grids the procedure becomes much more complicated. The use of WENO methods has been proposed [77], but remains a challenge.
- Explicit calculation of the distance to the interface. An example is [67], where an explicit 1D surface function is created, on the form $y = h(x)$ or $x = h(y)$ depending on the rotation of the surface. Then the distance from a point in space (x, y) to the surface function is calculated, using a Newton iteration proces.

A note about the contour plots produced in this thesis: We do not have a contour plotter which can produce high sub-element accuracy of the contours. The contour plotter is based on a triangulation of the nodes within each element. Within each element, linear interpolation between the nodes is used, giving a number of straight lines as contours, as in Figure 5.28, where the +-marker marks the line end points. Hence, it is only $\mathcal{O}((\Delta x/n)^2)$ order accurate, n being the degree of the

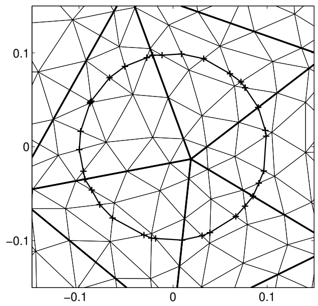


Figure 5.28: Contour plot accuracy

polynomial space P^n.

[1]Essentially Non-Oscillatory

Chapter 6

Numerical Tests

I can show you that the art of calculation has to do with odd and even numbers in their numerical relations to themselves and to each other.
— Plato 427–347 BC

In the following we shall present a few examples validating the performance of the proposed schemes.

The examples are intended for validation of the schemes, not for measuring of computational performance and accuracy. Thus we have not made any attempt to optimize the meshes and elements, which in all tests are of same size in the entire domain. However, in all tests there are regions which could be resolved using much larger elements, effectively decreasing the total number of elements and the computing time, and still maintaining accuracy.

All test have been carried out in Matlab, either on a SUN ultra-SPARCIII 1000Mhz system or a Intel PentiumIIIm 1200Mhz laptop. During implementation and testing, focus has been on flexibility to add new features and components, and not computational aspects as optimal memory usage and time efficiency. Although many operations in Matlab are based on optimized BLAS routines, there are still parts of the implementation that runs very inefficiently, taking much more time than it would in an optimized implementation. We will therefore not report any timings, and we will not compare time-accuracy-efficiency of the methods here with other existing methods.

The Matlab implementation is based on the USEMe package, [74]. The USEMe package handles the complicated logic in loading the mesh and setting up the standard operators.

6.1 Zalesaks Problem

Zalesak [76] proposed a test to evaluate how well a method transports an interface. The Zalesak disk is designed to mimic many of the interface configurations which may appear in a simulation: It has smooth round areas, sharp corners, and interfaces located close to each other. Testing on the Zalesak disk should therefore give a good idea of the performance of the method. The Zalesak disk, including the mesh which we will use, is depicted in Fig. 6.1. The disc is rotated one revolution

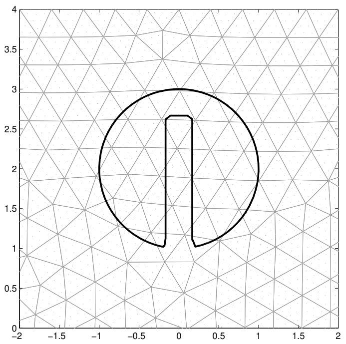


Figure 6.1: Zalesaks problem, including underlying mesh. Entire domain is $[-4;4]\times[-4,4]$ hence only a part of the domain is shown

around the center of the domain using

$$\phi_t - \mathbf{u}\cdot\nabla\phi = 0, \qquad \mathbf{u} = (-\pi y, \pi x)\ . \tag{6.1}$$

We shall use the fourth order explicit Runge Kutta scheme of [10] for time stepping Equation (6.1). The rotation is a shape preserving transformation, hence reinitialization is not necessary. In each Runge Kutta stage we will use a small amount of filtering to control oscillations coming from the discontinuities. Figure

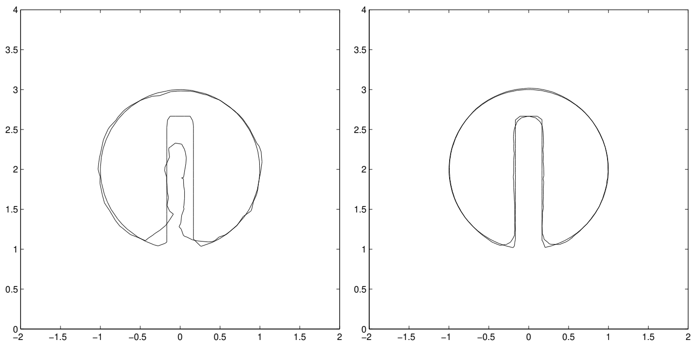


Figure 6.2: Zalesaks problem, solution after one revolution for 3rd (left) and 5th (right) order elements.

6.2 shows the initial disk and the disk after one full revolution using 3rd and 5th order elements respectively. The mesh has 894 elements. For comparison with other methods, counting the number of unknown, the 3rd order element has 10 and the 5th has 21 unknowns, in total 8940 and 18774 unknowns respectively, corresponding approximately to rectangular meshes of size 95x95 and 137x137.

Results using the 5th order elements show that the DG method can evolve the Zalesak disk well and solve it accurately without the need for any special techniques or tricks. Note that the width of the gap through the center of the circle is less than the side length of an element, and the disk itself is only resolved with about 30 elements, leading to a relatively coarse mesh.

Results using 3rd order elements show a typical degradation of the solution, when the DG method have difficulties representing the smallest scales of the level set. This may, however, be improved by using techniques presented in [66], designed to improve on the area conservation properties of the reinitialization.

The DG method works very well where the level set is smooth, on the circular part of the disk. Where the surface has sharp corners, the DG method smoothes the corner. In general we cannot expect high order of accuracy at such corners for the DG method, since the corners involve a discontinuity in the gradient of the level set, and high order cannot handle discontinuities well. On the other hand will surface tension quickly smoothen out such corners, and make sure that they will not reappear, at least for well resolved problems. Finally, the DG method is less accurate where two surfaces are very close, in the gap of the disc. This is again due to a discontinuity in the gradient of the level set, coming from the construction of the level set as a signed distance function.

Compared to lower order methods using a 137x137 mesh, the results are excellent, [31, 59, 66, 71, 76]

6.2 Standing Wave

The standing wave example is a gravity driven flow. In a box of size 1×2, the water in rest fills half the box, and the initial surface is set to $1+0.2\sin(\pi x)$, see Fig. 6.3. Densities ratio used are true values for water and air 1000:1, while the viscosity for both fluids are set to 1. Slip boundary conditions, i.e., no stress conditions [5], are

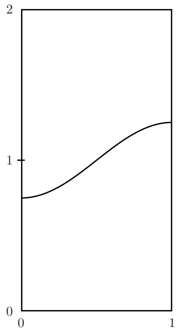


Figure 6.3: Standing Wave Initial Conditions

set at all boundaries, apart from at the 4 corners, where homogeneous Dirichlet conditions are used. Velocities are initially set to zero, and at time $t = 0$ the water is "released" and starts to move from side to side. Due to viscosity, the wave-height will decrease. The integration is carried out until $T = 2$, where the wave has moved forth and back almost 2 times.

The problem was solved on a grid having 62 elements of order 4. Figure 6.4 (left) shows the area of the water over time for different time steps. It shows that mass is quite well conserved even for a longer integration time on this very coarse grid. Figure 6.4 (right) shows the energy over time for different time steps. Energy is lost due to viscosity. If integrating once with very small time step and defining

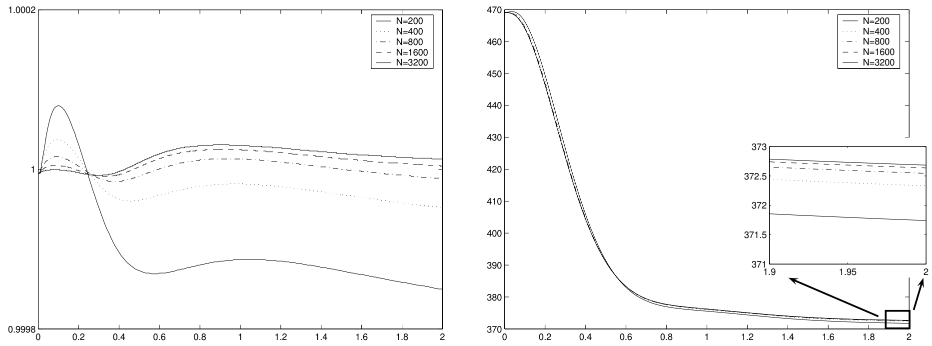


Figure 6.4: Standing Wave mass conservation (left) and energy (right) as a function of the timestep, $\Delta = 2/N$.

that as the true solution, the energy converges toward the true solution in $\mathcal{O}(\Delta t^{3/2})$ for a 2nd order BDF/Adams combination.

6.3 Wobbling Drop

The wobbling bubble test displays an example of a surface tension driven flow. A bubble formed initially as an ellipse and without a gravitational force, will oscillate due to the difference in curvature and thereby surface tension.

The test is set up in the domain $(x, y) \in [-\frac{1}{2}; \frac{1}{2}] \times [-\frac{1}{2}; \frac{1}{2}]$. Initial and boundary conditions are

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{0}, \tag{6.2a}$$

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{0}, \quad \text{on } \partial\Omega \tag{6.2b}$$

The surface is initially an ellipse with semi axis 0.2 and $\frac{\sqrt{2}}{10}$. Density ratio is

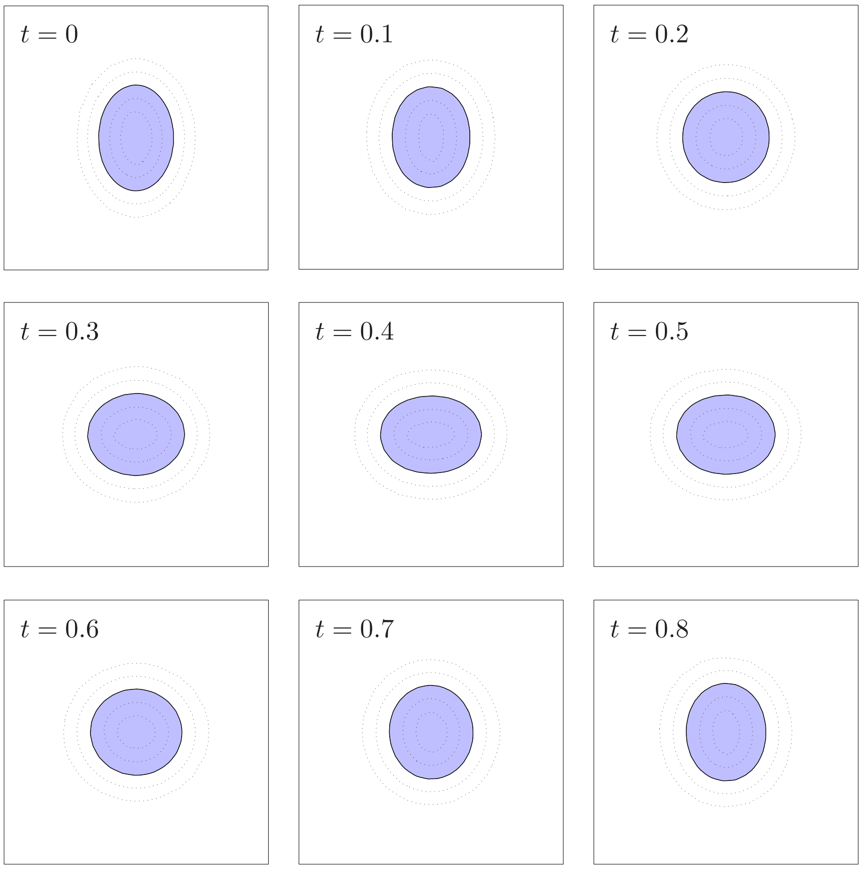


Figure 6.5: Surface at times $t = 0, 0.1, ..., 0.8$. Dotted lines are level set contours at $-0.1, -0.05, ..., 0.1$

1000:10 while viscosity ratio is $\frac{1}{2} : \frac{1}{2}$. Using the bubble rest diameter $L = 0.2$, $U = 0.1$, $\rho_l = 1000$, $\mu_l = 0.5$ gives a Reynolds number of $Re = 40$. Surface tension is of size $\sigma = 50$ giving a Weber number of $We = 0.4$.

The problem is solved on a mesh having 894 elements of order 3. The level set uses $\epsilon = 0.075$, which is a bit bigger than the side-length of the biggest element.

Figure 6.5 shows 9 still pictures of the wobbling bubble over about one period. Figure 6.6 shows the y-diameter of the bubble as a function of time. The diameter decreasing due to viscosity. The right plot shows the relative area as a function of time, and shows that the mass loss is about 3.5% after time $T = 2$.

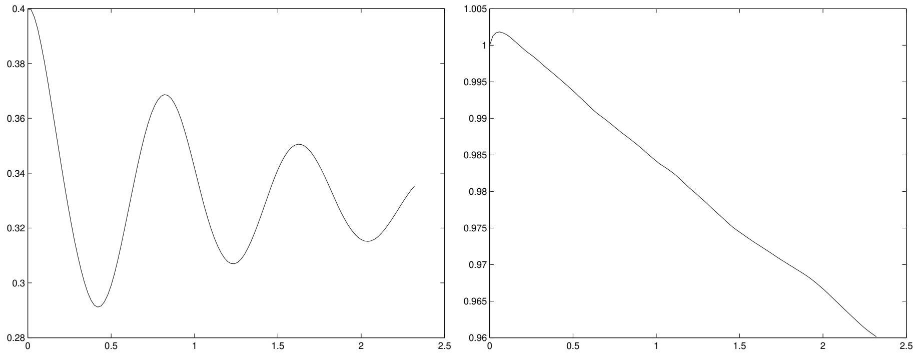


Figure 6.6: Bubble diameter (left) and area (right)

The mass loss is primarily due to effects depending on the surface thickness, described by the level set parameter ϵ. Decreasing ϵ would also decrease the mass loss. However, ϵ should match the element sizes, so we would need to increase the resolution.

Nevertheless, it illustrates one of the concerns often mentioned regarding the level set formulation, that mass is not conserved within the scheme. Mass loss will happen also at better resolution, but then at a smaller scale. As mentioned before, then techniques in [66] may be included to improve on the mass conservation properties.

It also illustrates one of the strengths of the DG method for the level set formulation, namely the ability to maintain shapes. Through the entire simulation, the elliptic interface is very nicely resolved. The resolution of 894 elements of order 3 gives in total 8940 unknowns, corresponding to a equidistant mesh of about 95×95 mesh-points. Hence the resolution is again very coarse.

6.4 Falling Droplet

The Falling droplet is another example where surface tension plays an important role. The test setup is the same as for the wobbling droplet, though a small amount of gravity has been added. The amount of gravity is small compared to surface tension. When the droplet hits the bottom wall, surface tension will act like rubber from a balloon wrapped around the water, trying to maintain a circular structure. The result is that the droplets bounces at the bottom wall. Figure 6.5 shows 9 still

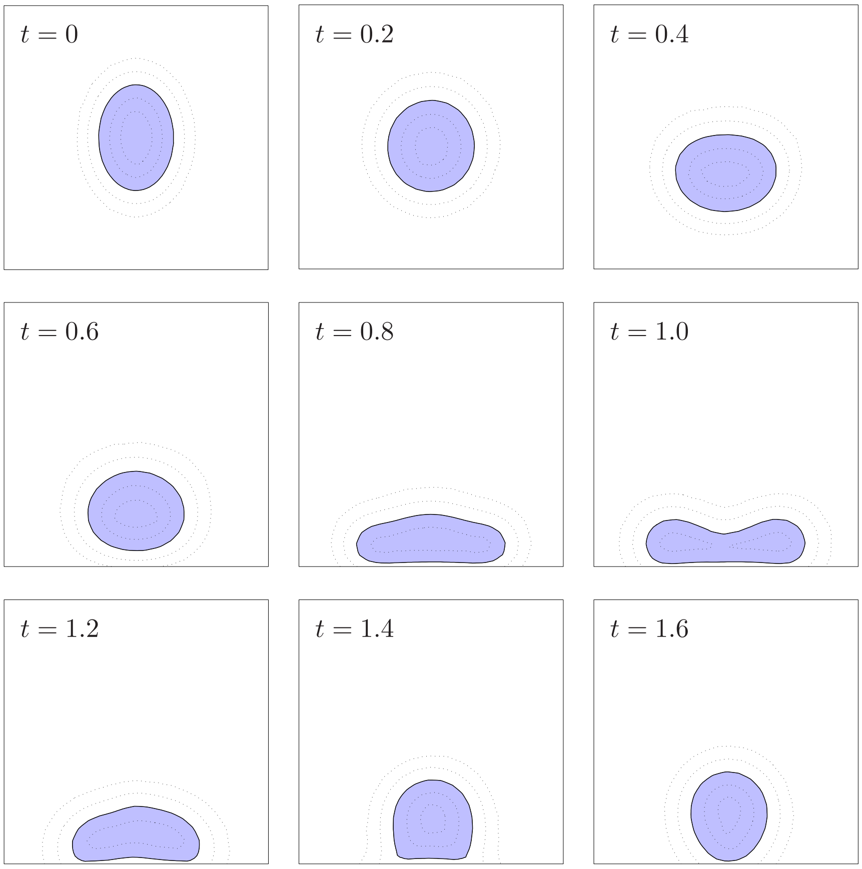


Figure 6.7: Surface at times $t = 0, 0.2, ..., 1.6$. Dotted lines are level set contours at $-0.1, -0.05, ..., 0.1$

pictures of the falling droplet. Due to viscosity, energy dissipates and the droplet does not completely lift from the bottom wall. However, increasing the resolution, it is possible to decrease the viscosity, and it should really bounce.

Notice, even though the initial conditions are symmetric, then the droplet is not completely symmetric during its fall. The mesh is not symmetric. Had we used a symmetric mesh, then symmetry could have been preserved. The surface tension makes the system quite stiff and difficult to solve, hence small numerical errors grows to visible asymmetries. It will slowly disappear when increasing the resolution.

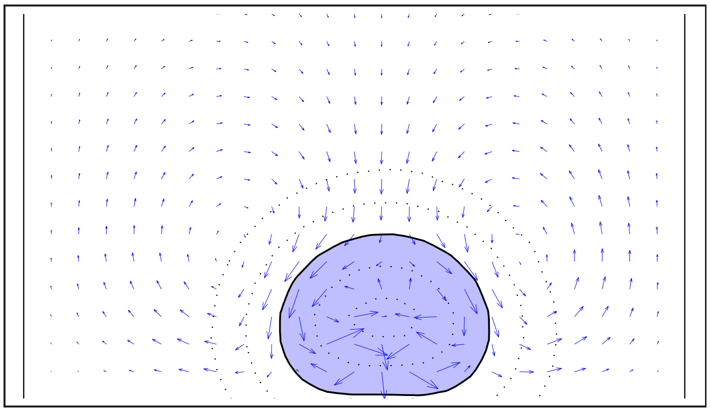


Figure 6.8: Zoom-in at droplet at time $t = 3.6$, only about 70% mass left

During this simulation, a large amount of mass is lost. Figure 6.8 shows the droplet and the flow field at time $t = 3.6$, where the droplet does not move anymore. Hence the flow is a steady state flow. At steady state, the flow should be zero everywhere, the force from gravity should balance the surface tension. But due to the interface thickness of $\epsilon = 0.075$, then the density changes from 1000 to 10 within ± 0.075 from the interface, i.e. basically within the outer dotted lines in the figure. This density difference drives the flow in the figure. And this flow slowly decreases the size of the droplet. This is again a problem that will diminish when resolution is increased and interface thickness decreased.

6.5 Flow Generated Surface Waves

This example displays generation of surface waves by an obstacle below the surface. The global domain is $(x, y) \in [-1; 1] \times [-\frac{1}{2}; \frac{1}{2}]$ having in the middle a rigid box of size 0.15×0.2. At the left boundary we shall apply a constant inflow having a constant water height, and we wish to explore what happens when the flow passes the rigid box. We shall use the following initial and boundary conditions

$$\mathbf{u}(\mathbf{x}, 0) = (1, 0), \tag{6.3a}$$

$$\mathbf{u}(\mathbf{x}, t) = (1, 0), \quad \text{at inflow, top and bottom boundaries} \tag{6.3b}$$

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{0}, \quad \text{on box boundaries} \tag{6.3c}$$

$$(\mathbf{n} \cdot \nabla)\mathbf{u}(\mathbf{x}, t) = \mathbf{0}, \quad \text{at outflow boundary.} \tag{6.3d}$$

Surface is initially at rest at height $\frac{1}{4}$, which is 0.15 higher than the top boundary of the box. At inflow the surface is set at height $\frac{1}{4}$ for all time. Density ratio is 1000:10 while viscosity ratio is 1:1. Gravity is set to $g = 10$. Using the box height in $L = 0.2$, the inflow condition in $U = 1$, $\rho_l = 1000$, and $\mu_l = 1$ gives a Reynolds number of $Re = 200$. The solution is time-stepped using a $\Delta t = 0.025$ to time $t_{\text{final}} = 8$ using the 2nd order BDF method. For the level set, we use $\epsilon = 0.15$, and

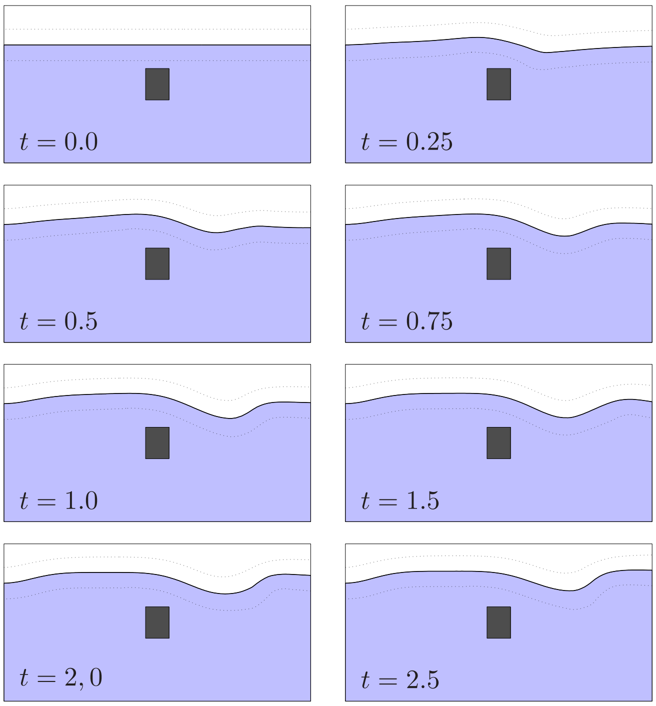


Figure 6.9: Surface at times $t = 0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5$

we will reinitialize after each step to $\tau = 0.0075$. Surface tension is neglected. The problem is solved on a mesh having 304 elements of order 5, the mesh can be seen in Appendix D.

Figure 6.9 shows 6 still pictures of the surface, while the Figure 6.10 shows 2 still pictures of the flow to the right of the obstacle, where streamlines has been added. There is no steady state solution. The surface will oscillate due to the

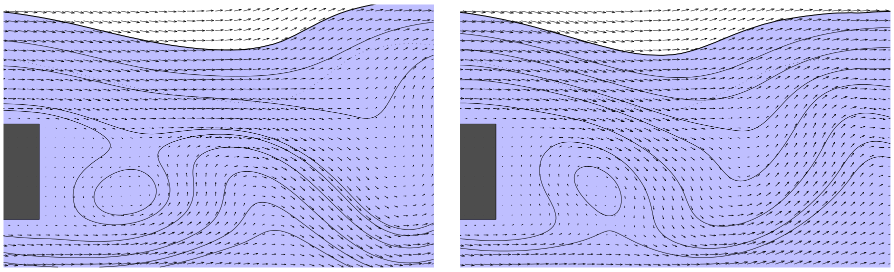


Figure 6.10: Detailed flow with streamlines behind box at time $t = 3.5$ (left) and $t = 4.0$ (right)

shedding of Von Karman vortices. The amplitude of the surface oscillations is not big, hence difficult to see in Figure 6.9, however the shedding of the vortices are very clear in Figure 6.10. We may estimate the period to twice the time between the two plots in Figure 6.10, giving a period of approximately one time unit, $T = 1$.

We can calculate the force on the box by integrating up the pressure at the box boundaries. In Figure 6.11, the force is split up into its x and y component. The

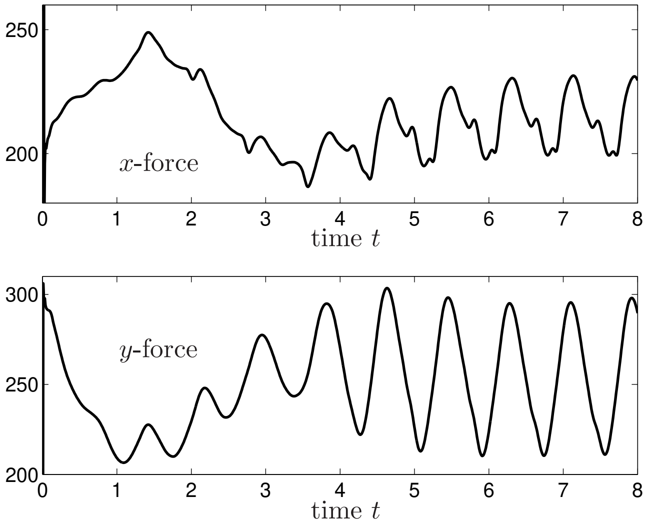


Figure 6.11: Force on box in x and y direction as a function of time

y component of the force corresponds to the force if the box had been massless. However also gravity pulls on the box, hence the actual lift applied to the box will be less, and depend on its weight. Nevertheless, the x component is the actual force applied to the box. We see that after a transition phase of about 4 time units, the flow enters a quasi-stationary state, where we can read the period to be slightly less than one time unit.

The example demonstrate the ability to add a structure and to calculate the flow around the structure and the load on the structure. The mesh is very coarse, and viscosity needs to be large in order for the system to be solvable. Even with this very coarse grid, we are able to model the unsteady flow behind the box, and the oscillating forces applied to the box by the flow.

Chapter 7

Discussion

Never hold discussions with the monkey when the organ grinder is in the room.
— Sir Winston Churchill 1874–1965

We have discussed the development of level set based discontinuous Galerkin methods as an approach to the modeling of the incompressible Navier-Stokes equations with free surfaces.

We have utilized a nodal high-order discontinuous Galerkin discretization on a fully unstructured grid.

- The fully unstructured approach makes it possible to handle very complex geometry, and to model a great variety of real life structures. This has been an important criteria, which is fulfilled. In practice the computer hardware and the time available will limit the complexity of problems that are solvable. However, being fully unstructured, it is possible to resolve complex regions well with many small elements, and regions where not much is happening can be resolved with few and larger elements, typically at inflow and outflow boundaries, thereby reducing the size of the system to solve.
- The geometric flexibility works hand in hand with the high-order setup: High order methods will resolve a wave well with only few elements per wavelength, hence at e.g. inflow boundaries we can make the element size of same order as the wavelength of the incoming waves, and we may still have sufficient accuracy.
- The nodal setup is convenient, since we are working with the physical variables at specified points in the domain. We can directly compare values and solutions, without requiring transformations between basis functions.
- High-order methods as the DG method result in more dense and in some cases fully dense matrix-vector and matrix-matrix operations. The computer

hardware of today has a strictly hierarchical memory structure, from disk and RAM over 2 to 3 levels of cache before reaching the CPU. Dense systems can take advantage of the memory structure and gain a much higher throughput than sparse systems, hence even in cases where the number of floating point operations are comparable in a dense setup compared to a similar sparse setup with same resolution, the dense version may finish many times faster than the sparse. We must include this point in our discussion when evaluating the method, since the important issue is not how many floating points operations are needed for a simulation, but how long time it actually takes to finish. This is not addressed in this thesis, since we do not yet have a fully optimized solver ready for these kind of tests.

The discontinuous approach gives a number of extra degrees of freedom compared to continuous spectral element methods, which can be used to help stabilize the system, e.g. by applying a diffusive numerical flux between elements.

- The DG method are suited and works well for wave dominated problems. It is here shown to work for the incompressible Navier Stokes, and also for the level set treatment. This is partly due to one of the major strengths of the DG method presented here: It is very versatile. For a problem where the main part is solved well by the DG method, and having a subproblem for which the DG method is not the optimal, the DG method can usually be twisted and tweaked to produce a fair result also for the subproblem. This is the case for the level set and especially the level set reinitialization. Hence we do not need to apply another scheme to the subproblem, and cope with any problems arising in the “no-man’s land” between the two methods.

We are solving the problem as a two-fluid problem with the interface described by the level set.

- The level set was introduced to enable the robust and simple representation of the free surface. We have discussed appropriate boundary conditions for the level set at inflow and outflow boundaries. A few tests confirm the accuracy, robustness, and versatility of the proposed scheme.
- We are solving both for the water part and the air part. The influence of the air part to the flow of the water and the load on the structures is small, and it can in many cases be neglected. Hence the solution of the air part impose extra work, which strictly is not necessary. If possible, we must make sure that the air part is kept as small as possible to minimize unnecessary computational work.
- As mentioned earlier, the techniques presented here works for other pairs of non-mixing fluids, e.g. in the case of oil and water. Then both fluids are equally important and the above point is no longer valid. The proposed method will probably work even better for an oil-water case than the water-air, since one of the delicate problems with water-air case is the big difference

in density and viscosity.

- The level set formulation implies some amount of mass loss during simulations, the amount depending on the resolution and the problem solved. It is possible to include techniques to improve on the mass conservation properties. The techniques presented in [66] includes calculation of the volume of the water domain, and applies strategies to locally conserve mass in the reinitialization equation. Other possibilities are to combine a Volume of Fluid and a level set formulation [65], using the mass conservation properties of the VOF method and interface derivatives from the level set. This is also possible using the DG method, however the coupling of the VOF and level set is not as straightforward as for other methods.

  There will be cases where the mass loss of the level set method is a problem. For example are mass loss problems common for the sloshing wave in a tank: The natural frequency of the sloshing wave depend on the amount of water, hence mass conservation is important. Even VOF methods show problems in this setup [52]. Whether increasing the resolution will solve the problem of mass loss satisfactory, is a subject for further investigation. However, there will also be problems, where the mass loss is not important, and the level set formulation will work well.

- The DG method together with the level set formulation has a superb ability to preserve shapes of smooth interfaces. Other methods as the VOF method will have errors coming from the fact that shape is distorted. However only the level set method can provide the high order sub-element accuracy of the surface position, which the DG method can exploit.

We have applied a velocity projection scheme in combination with a semi-implicit approach in time to advance the unsteady equations. As an alternative we propose the use of a semi-implicit spectral deferred correction projection (SISDC) formulation which has a number of attractive properties:

- The SISDC method is very flexible, based on low order methods and capable of producing very high order of accuracy. The flexibility consist of the ability to split and decouple systems using any low order method, and then correct the errors iteratively.

- It remains unknown how to use it in finite domains without impacting the accuracy adversely.

  Firstly, we do not know how to apply boundary conditions for the pressure projection in order to retain accuracy, a problem also reported in [48, 50].

  Secondly, the choice of how to apply boundary conditions for the velocity is not trivial. For Runge-Kutta methods, it has been shown that applying exact boundary conditions in the implicit solve of each stage, will result in a order reduction [11, 60]. For many cases, solutions have been proposed to solve the order reduction problem for Runge Kutta methods [1, 55]. Similar problems arise for the SISDC method, producing boundary layer errors. For some problems a solution has been found for the fully implicit deferred correction schemes [44]. However, no general approach has yet been found.

- The resulting general time scheme is based on a BDF formula. This is an old and well known technique, providing up until $2\frac{1}{2}$ order of accuracy. This is a fair result, however our hope has been to achieve better.

7.1 Open Questions and Further Work

The present work suggests a number of open questions, possible improvements and ideas for further work.

7.1.1 Open Questions

The correct treatment of boundaries in the SISDC approach is one of interest as the formulation appears to have the potential to reach higher accuracy than with standard projection methods.

Index reducing the incompressible Navier Stokes and solving the underlying system of ODEs has to our knowledge not been tried. Whether it is possible and how efficient it may be, is an open question. Though, solving the underlying ODE for the pressure may remove some of the stability problems which the projection techniques suffer from. It would be interesting to try the SISDC approach on the underlying ODE.

Furthermore, the introduction of the level set raises a number of questions in relation to its treatment around no-slip boundaries and multiple connected surfaces. Since there is zero velocity on no-slip boundaries, the zero level set will never intersect or move on a no-slip boundary, e.g., for a wave hitting the box, the zero level set contour would wrap around the box. Whether this "wrapping" is acceptable, or a heuristic should be applied for updating the level set in cells on no-slip boundaries newly filled with/emptied for water, or if slip conditions are more appropriate, is a topic for further investigation.

7.1.2 Further work

There are details in the work which has been determined using a heuristic trial-and-error approach. However a more thorough analysis or systematic test would give more insight into the properties of the method, and more optimal parameters may improve on accuracy or performance. This includes:

- The filter levels in the adaptive filtering process for the level set reinitialization. We would want to know how to determine the filter levels depending on the spatial discretization and the time stepping of the reinitialization.

- The streamline diffusion constant in the level set reinitialization. We would want to know how the constant should depend on the spatial discretization in order to make it as small as possible and still have the desired effect. Furthermore, as with the adaptive filter, we may also vary the diffusion constant depending on some smoothness criteria, and only apply the diffusion when needed.

Finally, the methods have not been tested against other methods for accuracy and computational efficiency. For the method to be attractive, it must at least compare with existing methods. This is an important task, including tests on realistic examples, validation of results against model experiments or other methods, and comparison of efficiency with existing methods.

The first step would be to implement the method in 2D in a high performance programming language, for example C. There already exists a C code, called USEMe[1] [74], handling the logics of grid setup, element connectivity, and creation of the standard operators. It exists in a version solving the compressible Navier Stokes. Hence it needs a Poisson solver, a Helmholtz solver and everything concerning the level set.

Three Dimensional Setup

As many effects in water wave modeling are three dimensional, it is also a wish to implement the presented methods in a 3D version.

Theory already exists for nodal distributions and operators for discontinuous Galerkin methods on tetrahedons . The description of 2D discontinuous Galerkin for conservation laws in Chapter 3 have already been extended to 3D. Actually, the description in Chapter 3 is fairly straightforward extendable to 3D, with only minor modifications. The actual implementation is of course more complicated due to now a 3D connectivity of elements. However, the USEMe code exist also in a 3D version in both Matlab and C, handling, as in 2D, the logics of grid setup, element connectivity, and creation of the standard operators. What is left to do is, again, to implement a Poisson/Helmholtz solver, and implement and test the level set methods. Hopefully it will work directly or with only minor modifications.

Parallel Setup

The modeling of free surface flows are, from a computational point of view, very expensive. The demand for more accurate results and more complicated domains by far surpass the computational power at hand, hence free surface flow simulations will keep pushing computers to their limit, and keep us employed some time yet. To gain the most computational power possible today, parallel computers must be considered. Hence the method should also be developed in a parallel version. The USEMe code exists in an MPI[2] version, designed for problems using explicit time

[1]Unstructured Spectral Element Methods.
[2]Message Passing Interface, used for passing messages in parallel environments.

stepping. To use it for modeling free surface flows, the biggest task is to implement a parallel Poisson/Helmholtz solver.

APPENDIX A

Nodes and Basis of standard element

Figure A.1 shows the numbering of the nodes of the standard triangular element. The three corner points are numbered first, then the 3 edges. Finally the interior nodes are numbered more or less arbitrary.

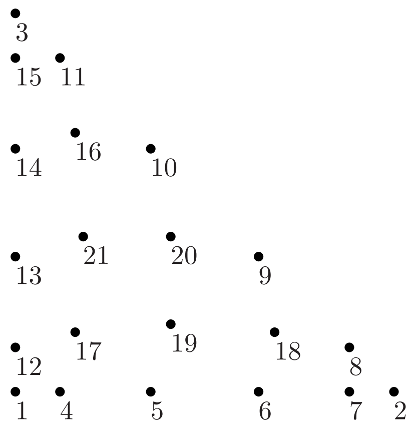


Figure A.1: Numbering of nodes for the standard triangular element, polynomail order 5

The number of nodes, and thereby the number of nodal basis functions depends on the order of the element. For the nth order element, we have:

$$u \in \mathsf{P}^n \quad \Rightarrow \quad \frac{(n+1)(n+2)}{2} \text{ nodes and basisfunctions}$$

In Figure A.2 we show 6 of the 21 basisfunctions from a 5th order element. Figure A.3 shows 8 of the 45 basis functions from a 8th order element.

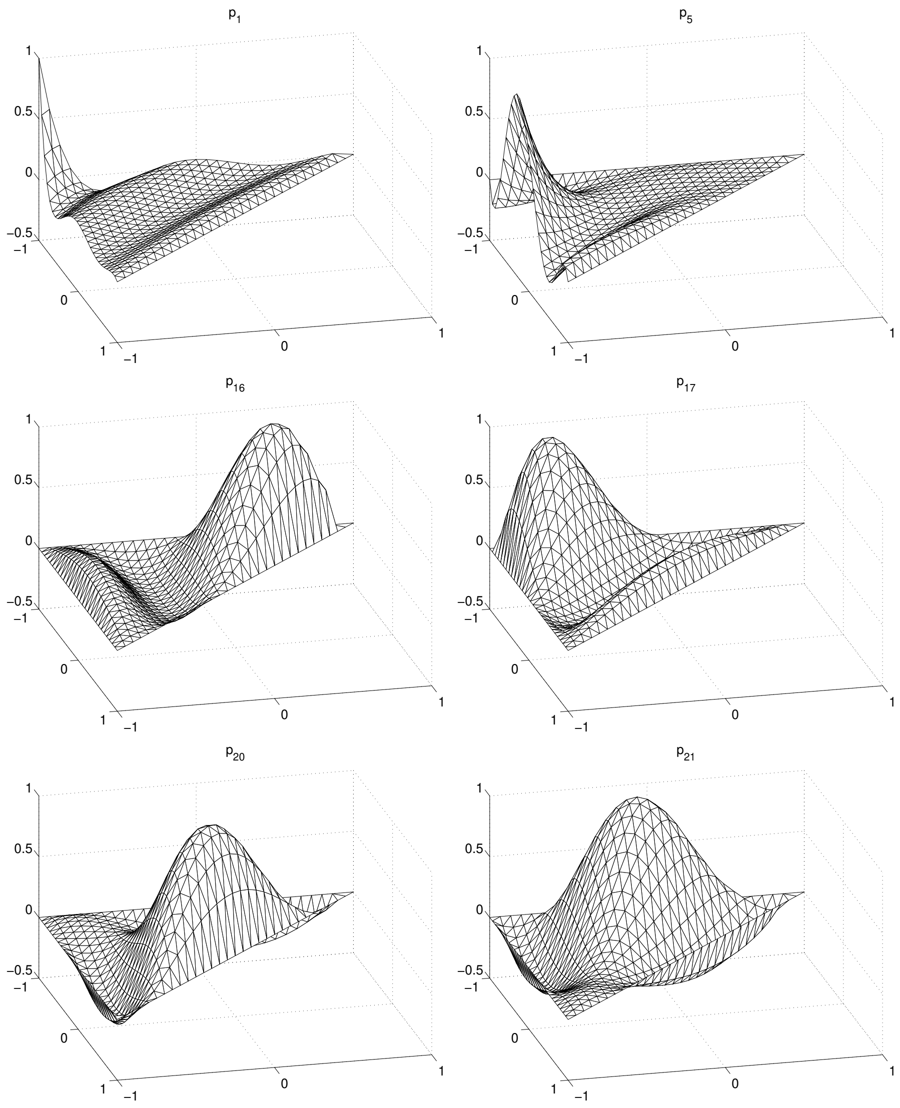


Figure A.2: 6 nodal basis functions for the standard triangular element of order 5

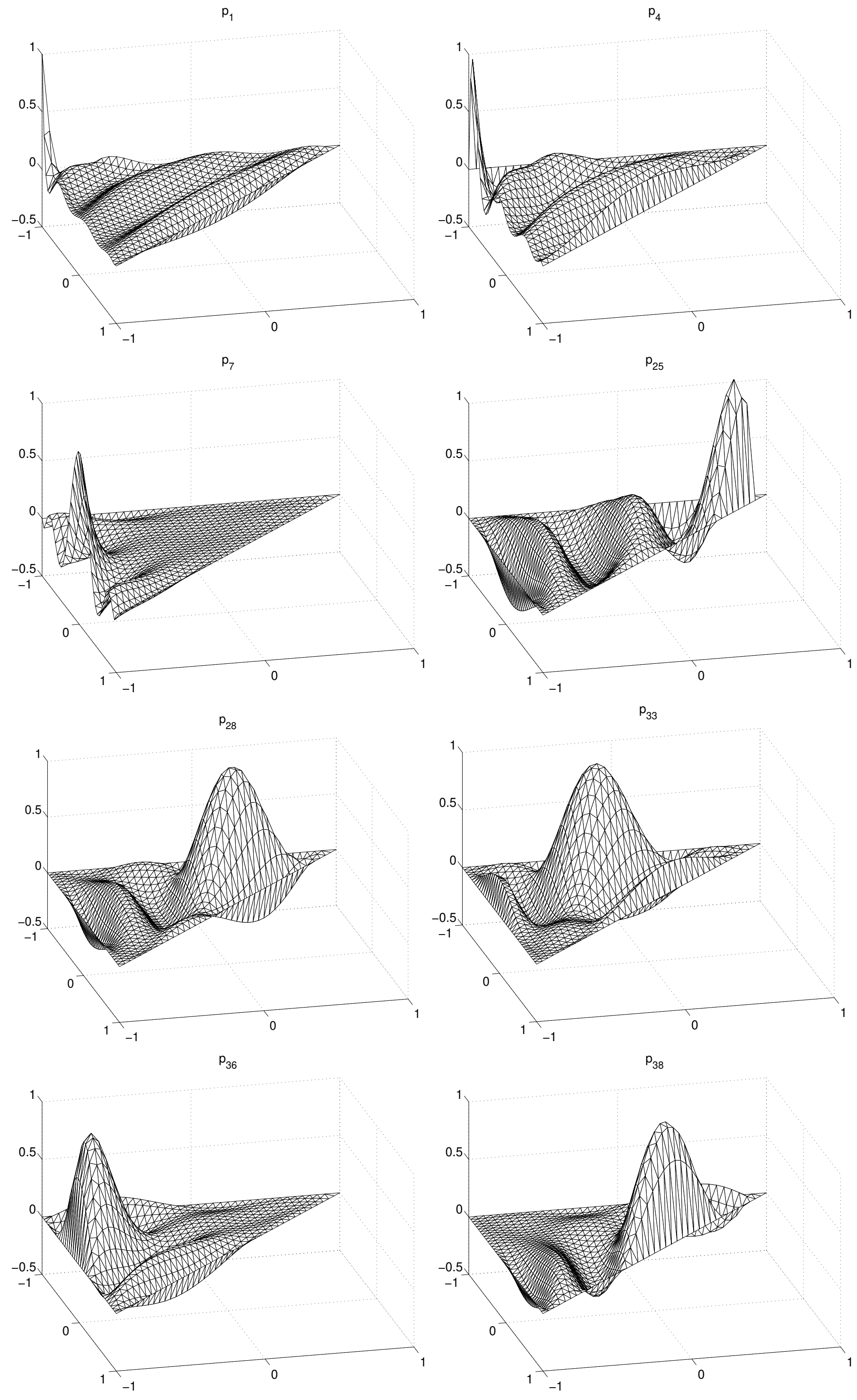


Figure A.3: Nodal basis functions for the standard triangular element of order 8

Appendix B

Order Plots for DAE Solvers

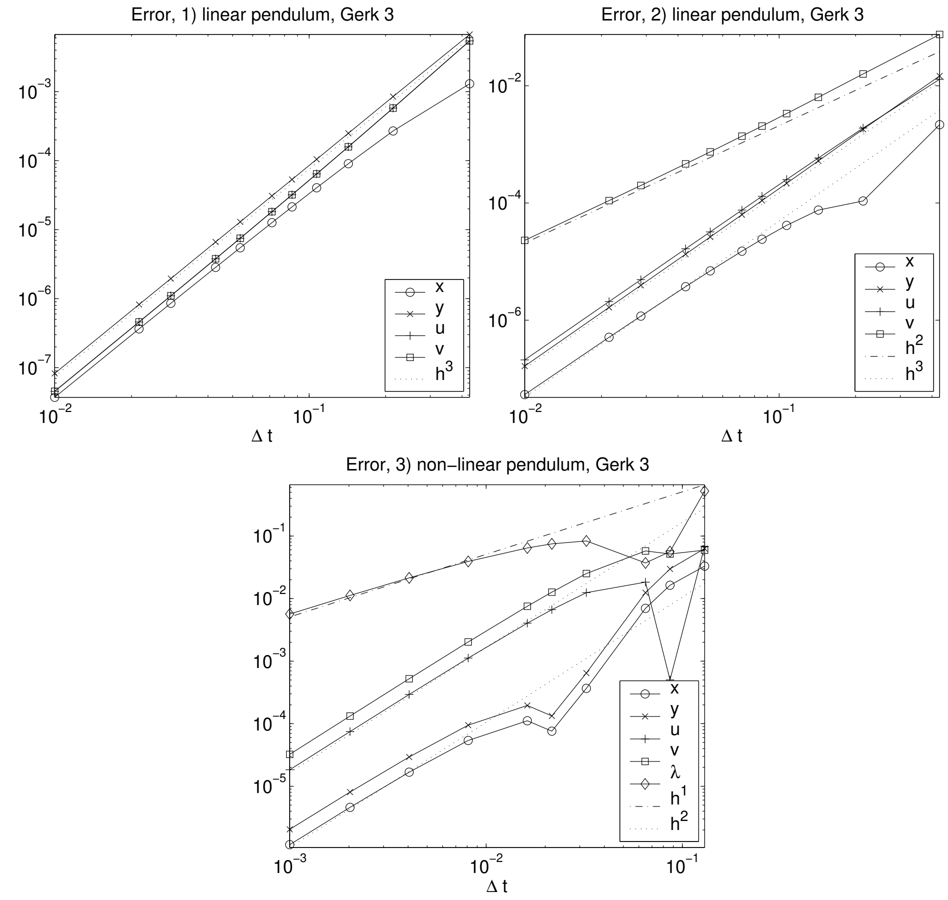


Figure B.1: Order results for the Gerk 3 method for all three DAE test problems. Note the dashed and dash-dotted line comparing the convergence with the expected order.

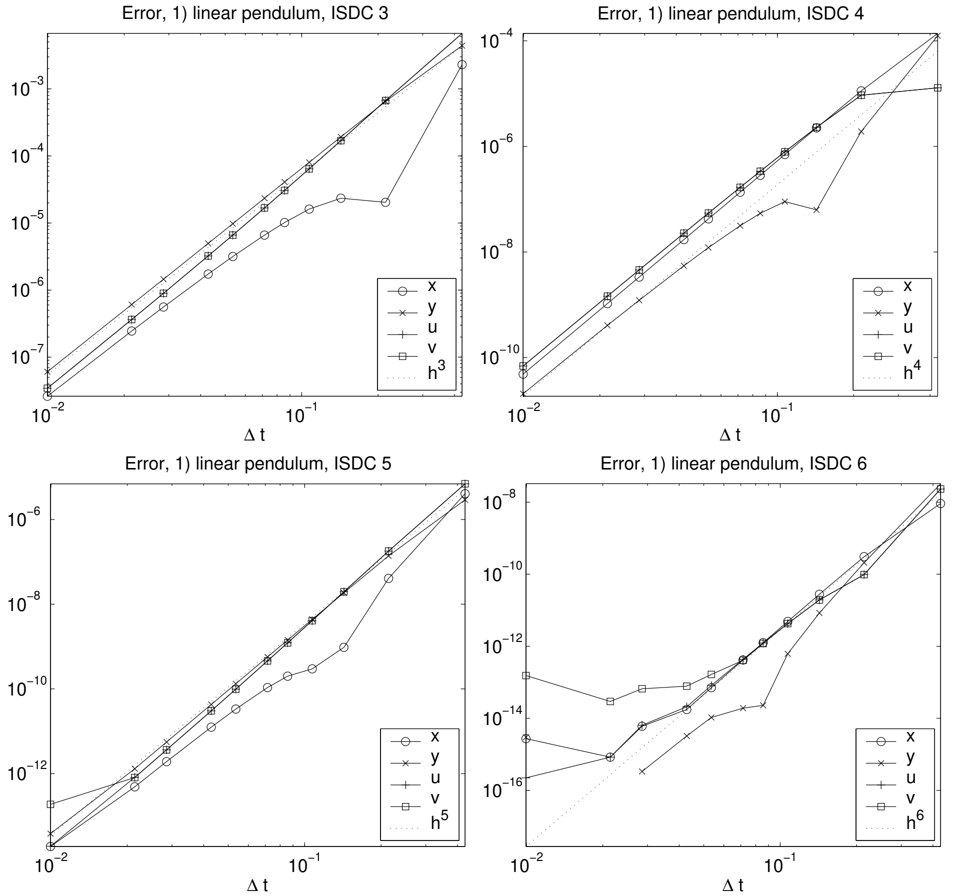


Figure B.2: Order results for the ISDC method for the linear pendulum 1) problem. Note the dashed line comparing the convergence with the expected order

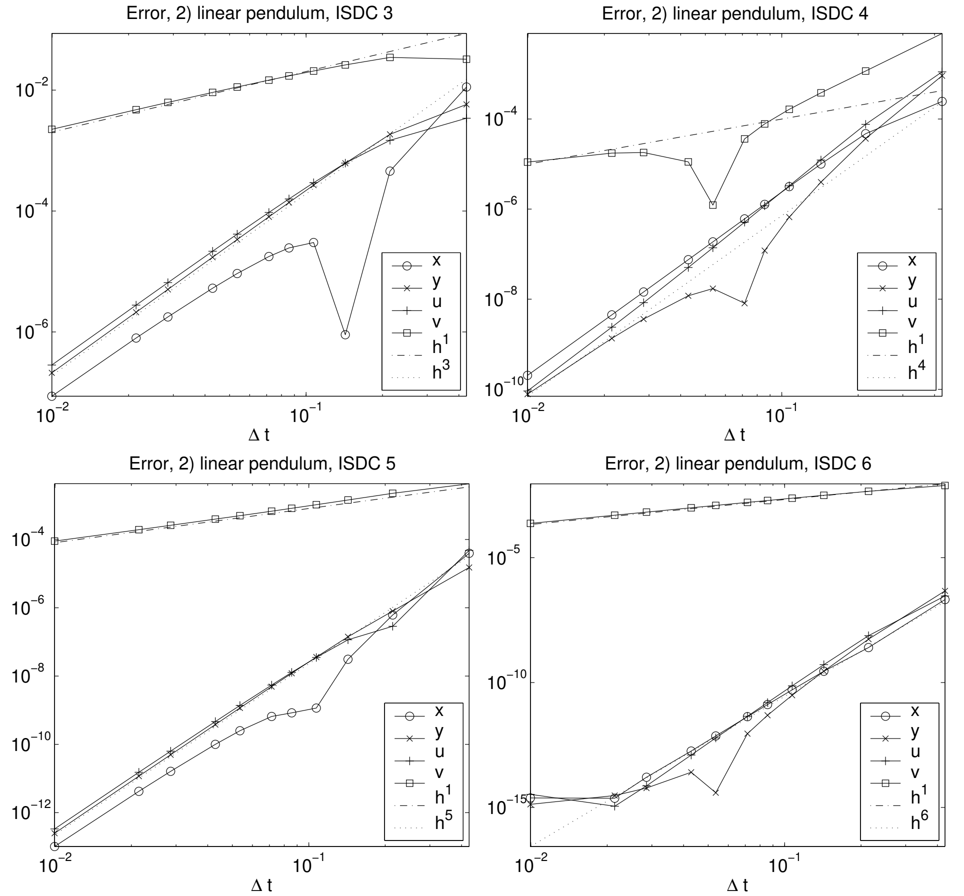


Figure B.3: Order results for the ISDC method for the linear pendulum 2) problem. Note the dashed and dash-dotted line indicating different orders.

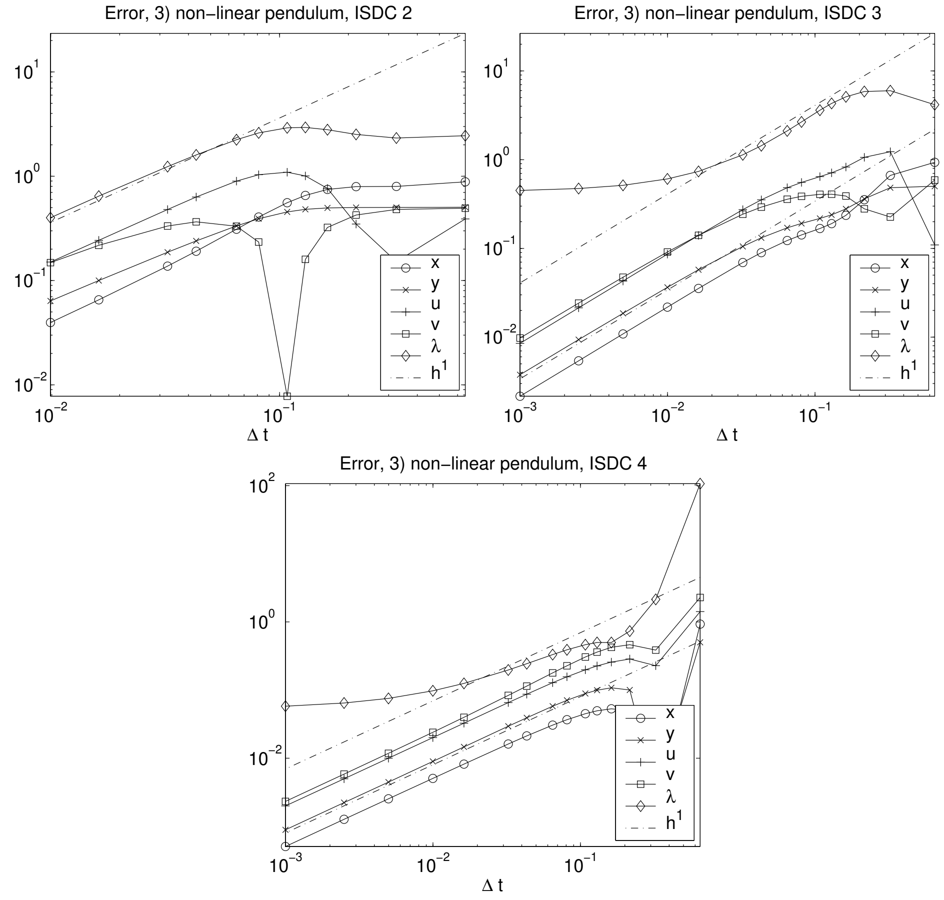


Figure B.4: Order results for the ISDC method for the non-linear pendulum 3) problem.

APPENDIX C

VOF figures

Figure C.2 and C.3 is from [52], where a Volume of Fluid method is tested on a simple convection problem. An initial circular interface is convected over a regular uniform mesh of size 200×100, using a constant velocity field to move the interface from the lower left corner to the upper right corner. The interface diameter is about 9 gridcells wide, the interior covers approximately 60 computational cells. Ideally the fluid fraction should be convected without changing the interface. As

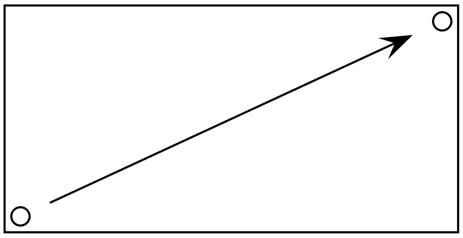


Figure C.1: Domain for convection problem in [52]

can be seen in Figure C.2 and C.3, this is not the case for any of the methods.

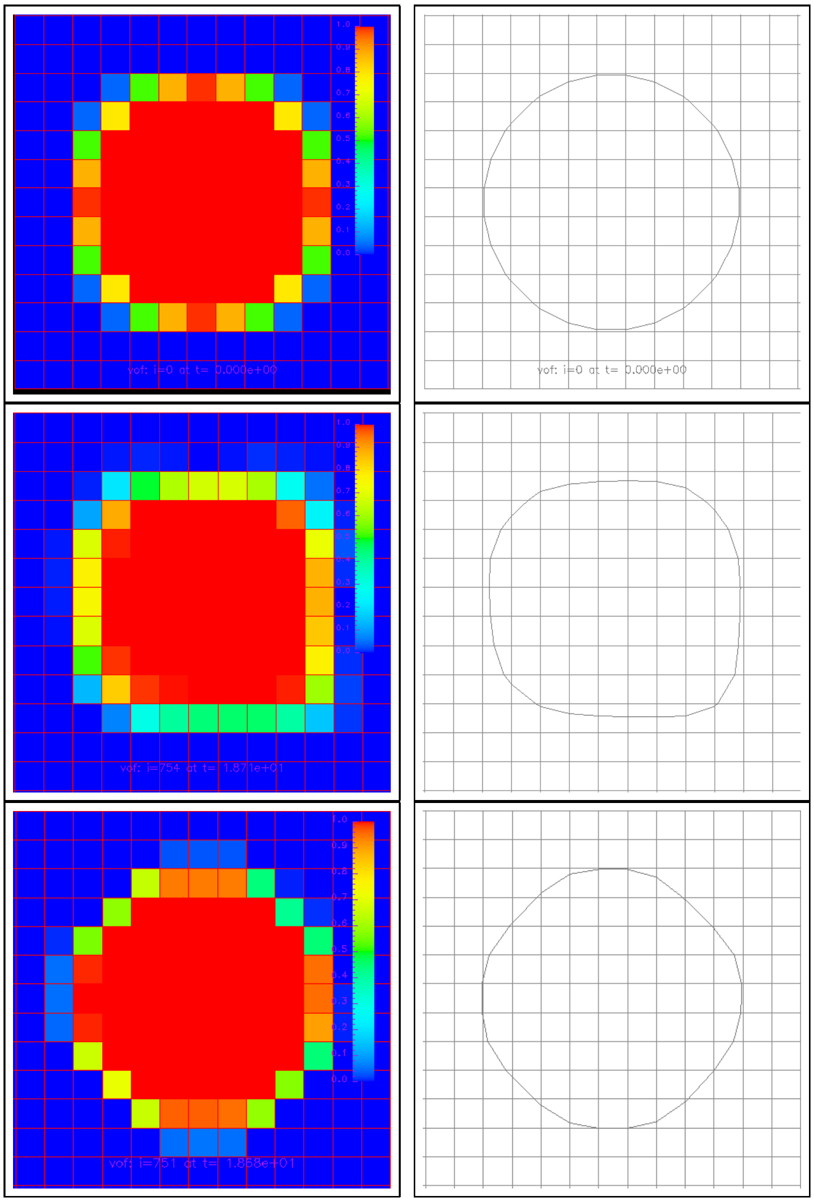


Figure C.2: Figures from [52], initial volume fraction (top), final volume fraction using method CICSAM (middle) and Hyper-C (bottom)

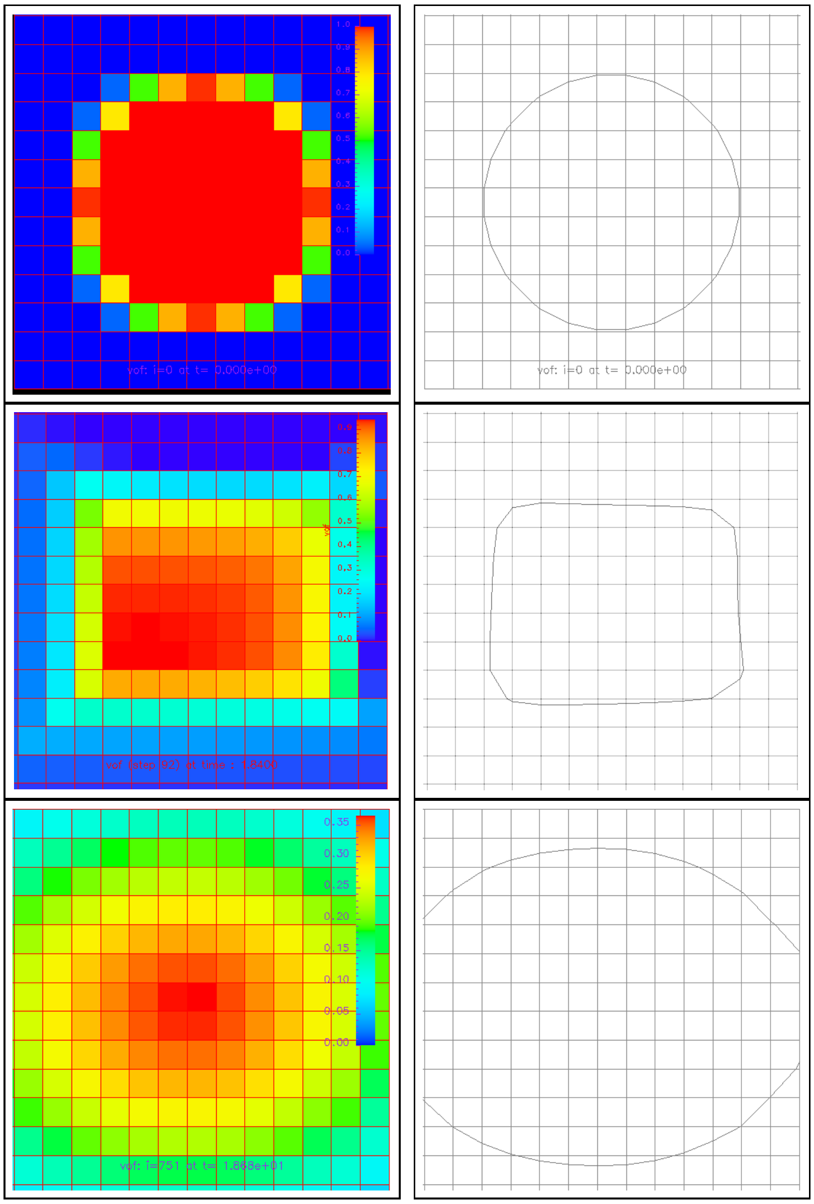


Figure C.3: Figures from [52], initial volume fraction (top), final volume fraction using method HRIC (middle) and UltimateQuickest (bottom)

APPENDIX D

Grids

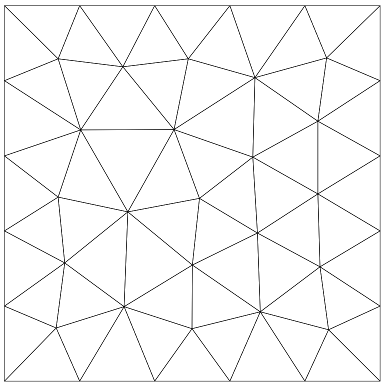


Figure D.1: Square grid, sidelenght $\Delta x = 0.2$, 62 elements

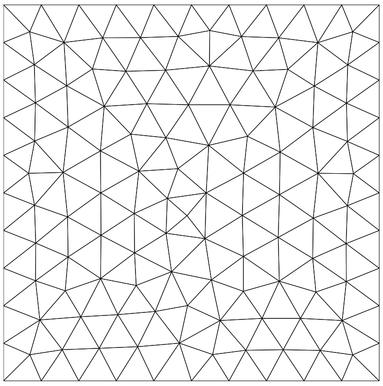


Figure D.2: Square grid, sidelenght $\Delta x = 0.1$, 226 elements

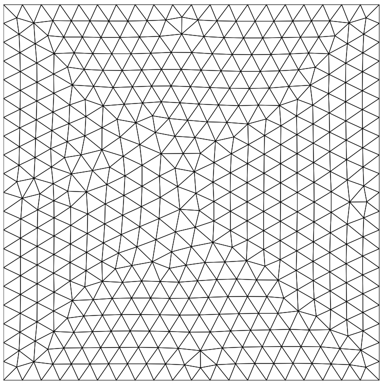


Figure D.3: Square grid, sidelenght $\Delta x = 0.05$, 894 elements

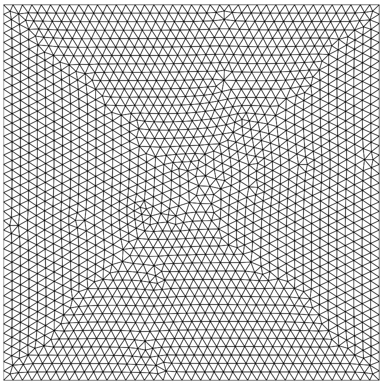


Figure D.4: Square grid, sidelenght $\Delta x = 0.025$, 3602 elements

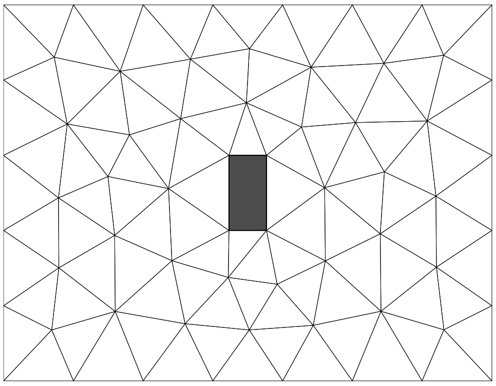


Figure D.5: Rectangular grid with box, sidelenght $\Delta x = 0.2$, 98 elements

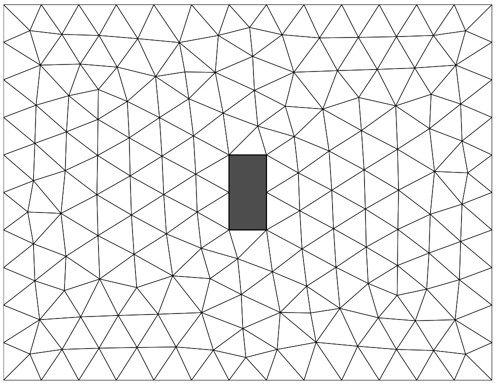


Figure D.6: Rectangular grid with box, sidelenght $\Delta x = 0.1$, 304 elements

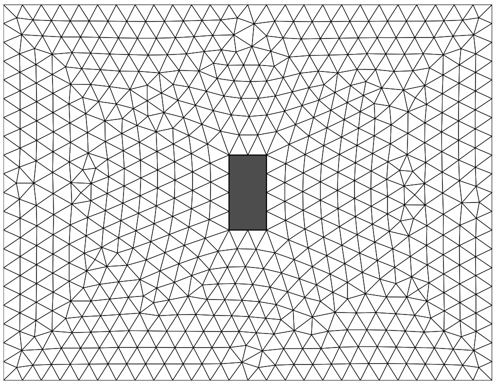


Figure D.7: Rectangular grid with box, sidelenght $\Delta x = 0.05$, 1202 elements

Bibliography

[1] Saul Abarbanel, David Gottlieb, and Mark H. Carpenter. On the removal of boundary errors caused by Runge-Kutta integration of nonlinear partial differential equations. *SIAM J. Sci. Comput.*, 17(3):777–782, 1996.

[2] J. D. Anderson. *Computational fluid dynamics - the basics with applications.* Mechanical Engineering Series. McGraw-Hill, Singapore, 1995.

[3] Douglas N. Arnold, Franco Brezzi, Bernardo Cockburn, and L. Donatella Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, 39(5):1749–1779 (electronic), 2001/02.

[4] Tariq D. Aslam. A level set algorithm for tracking discontinuities in hyperbolic conservation laws. II. Systems of equations. *J. Sci. Comput.*, 19(1-3):37–62, 2003. Special issue in honor of the sixtieth birthday of Stanley Osher.

[5] Eberhard Bänsch and Burkhard Höhn. Numerical treatment of the Navier-Stokes equations with slip boundary condition. *SIAM J. Sci. Comput.*, 21(6):2144–2162 (electronic), 2000.

[6] F. Bassi and S. Rebay. A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 131(2):267–279, 1997.

[7] G. K. Batchelor. *An introduction to fluid dynamics.* Cambridge Mathematical Library. Cambridge University Press, Cambridge, paperback edition, 2000.

[8] John B. Bell, Phillip Colella, and Harland M. Glaz. A second-order projection method for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 85(2):257–283, 1989.

[9] David L. Brown, Ricardo Cortez, and Michael L. Minion. Accurate projection methods for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 168(2):464–499, 2001.

[10] M. H. Carpenter and C. A. Kennedy. Fourth-order 2N-storage Runge-Kutta schemes. NASA Technical Memorandum 109112, NASA, Langley Reseach Center, 1994.

[11] Mark H. Carpenter, David Gottlieb, Saul Abarbanel, and Wai Sun Don. The theoretical accuracy of Runge-Kutta time discretizations for the initial-boundary value problem: a study of the boundary error. *SIAM J. Sci. Comput.*, 16(6):1241–1252, 1995.

[12] Paul Castillo. Performance of discontinuous Galerkin methods for elliptic PDEs. *SIAM J. Sci. Comput.*, 24(2):524–547 (electronic), 2002.

[13] Bernardo Cockburn, Guido Kanschat, and Dominik Schötzau. A locally conservative LDG method for the incompressible Navier-Stokes equations. Preprint PIMS-04-7, Pacific Institute for the Mathematical Sciences, 2004.

[14] Bernardo Cockburn and Chi-Wang Shu. The local discontinuous Galerkin method for time-dependent convection-diffusion systems. *SIAM J. Numer. Anal.*, 35(6):2440–2463 (electronic), 1998.

[15] Bernardo Cockburn and Chi-Wang Shu. The Runge-Kutta discontinuous Galerkin method for conservation laws. V. Multidimensional systems. *J. Comput. Phys.*, 141(2):199–224, 1998.

[16] Bernardo Cockburn and Chi-Wang Shu. Runge-Kutta discontinuous Galerkin methods for convection-dominated problems. *J. Sci. Comput.*, 16(3):173–261, 2001.

[17] M. O. Deville, P. F. Fischer, and E. H. Mund. *High-order methods for incompressible fluid flow*, volume 9 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2002.

[18] Moshe Dubiner. Spectral methods on triangles and other domains. *J. Sci. Comput.*, 6(4):345–390, 1991.

[19] Alok Dutt, Leslie Greengard, and Vladimir Rokhlin. Spectral deferred correction methods for ordinary differential equations. *BIT*, 40(2):241–266, 2000.

[20] Edda Eich. *Projizierende Mehrschrittverfahren zur numerischen Lösung von Bewegungsgleichingen technischer Mehrkörpersysteme mit Zwangsbedingungen und Unstetigkeiten*, volume 109 of *Fortschritt-Berichte VDI, Reihe 18: Mechanik/Bruchmechanik*. VDI Verlag, 1991.

[21] Olivier Faugeras and Renaud Keriven. Variational principles, surface evolution, PDE's, level set methods, and the stereo problem. *IEEE Trans. Image Process.*, 7(3):336–344, 1998.

[22] J. H. Ferziger and M. Perić. *Computational methods for fluid dynamics*. Springer-Verlag, Berlin, 3rd revised edition, 2002.

[23] D. Gottlieb and J. S. Hesthaven. Spectral methods for hyperbolic problems. *J. Comput. Appl. Math.*, 128(1-2):83–131, 2001.

[24] J. Grooss. Parallel elliptic PDE solver. Master's thesis, Informatics and Mathematical Modelling, IMM, Technical University of Denmark, 2001.

[25] J. Grooss and Hesthaven J. S. A level set discontinuous galerkin method for free surface flows. *Computer Methods in Applied Mechanics and Engineering*, submitted:xx–xx+30, 2005.

[26] J. L. Guermond and Jie Shen. A new class of truly consistent splitting schemes for incompressible flows. *J. Comput. Phys.*, 192(1):262–276, 2003.

[27] J. L. Guermond and Jie Shen. Velocity-correction projection methods for incompressible flows. *SIAM J. Numer. Anal.*, 41(1):112–134 (electronic), 2003.

[28] E. Hairer and G. Wanner. *Solving ordinary differential equations. II*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1996. Stiff and differential-algebraic problems.

[29] Ernst Hairer, Christian Lubich, and Michel Roche. *The numerical solution of differential-algebraic systems by Runge-Kutta methods*, volume 1409 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1989.

[30] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8:2182–2189, 1965.

[31] Dalton J. E. Harvie and David F. Fletcher. A new volume of fluid advection algorithm: the defined donating region scheme. *Internat. J. Numer. Methods Fluids*, 35(2):151–172, 2001.

[32] J. S. Hesthaven. From electrostatics to almost optimal nodal sets for polynomial interpolation in a simplex. *SIAM J. Numer. Anal.*, 35(2):655–676 (electronic), 1998.

[33] J. S. Hesthaven. A stable penalty method for the compressible Navier-Stokes equations. III. Multidimensional domain decomposition schemes. *SIAM J. Sci. Comput.*, 20(1):62–93 (electronic), 1998.

[34] J. S. Hesthaven and D. Gottlieb. Stable spectral methods for conservation laws on triangles with unstructured grids. *Comput. Methods Appl. Mech. Engrg.*, 175(3-4):361–381, 1999.

[35] J. S. Hesthaven and R. M. Kirby. Filtering in legendre spectral methods. *Math. Comp*, submitted (2004).

[36] J. S. Hesthaven and C. H. Teng. Stable spectral methods on tetrahedral elements. *SIAM J. Sci. Comput.*, 21(6):2352–2380 (electronic), 2000.

[37] J. S. Hesthaven and T. Warburton. Nodal high-order methods on unstructured grids. I. Time-domain solution of Maxwell's equations. *J. Comput. Phys.*, 181(1):186–221, 2002.

[38] C.W. Hirt and B.D. Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *J. of Computational Physics*, 39:201–225, 1981.

[39] Changqing Hu and Chi-Wang Shu. A discontinuous Galerkin finite element method for Hamilton-Jacobi equations. *SIAM J. Sci. Comput.*, 21(2):666–690 (electronic), 1999.

[40] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 1996.

[41] A. Kanevsky, M. H. Carpenter, and J. S. Hesthaven. Time-consistent filtering in spectral and spectral element methods. *draft*, 2004.

[42] George Em. Karniadakis, Moshe Israeli, and Steven A. Orszag. High-order splitting methods for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 97(2):414–443, 1991.

[43] Tom Koornwinder. Two-variable analogues of the classical orthogonal polynomials. In *Theory and application of special functions (Proc. Advanced Sem., Math. Res. Center, Univ. Wisconsin, Madison, Wis., 1975)*, pages 435–495. Math. Res. Center, Univ. Wisconsin, Publ. No. 35. Academic Press, New York, 1975.

[44] Wendy Kress and Bertil Gustafsson. Deferred correction methods for initial boundary value problems. In *Proceedings of the Fifth International Conference on Spectral and High Order Methods (ICOSAHOM-01) (Uppsala)*, volume 17, 1–4, pages 241–251, 2002.

[45] Jian-Guo Liu and Chi-Wang Shu. A high order discontinuous Galerking method for 2D incompressible flows. ICASE Report 99-27, NASA, Langley Reseach Center, 1999.

[46] Ravi Malladi and James A. Sethian. Level set methods for curvature flow, image enchancement, and shape recovery in medical images. In *Visualization and mathematics (Berlin-Dahlem, 1995)*, pages 329–345. Springer, Berlin, 1997.

[47] S. Mayer, A. Garapon, and L.S. Sørensen. A fractional step method for unsteady free surface flow with applications to non-linear wave dynamics. *International Journal for Numerical Methods in Fluids*, 28:293–315, 1998.

[48] Michael L. Minion. Higher-order semi-implicit projection methods. In *Numerical simulations of incompressible flows (Half Moon Bay, CA, 2001)*, pages 126–140. World Sci. Publishing, River Edge, NJ, 2003.

[49] Michael L. Minion. Semi-implicit spetral deferred correction methods for ordinary differential equations. *Comm. Math. Sci.*, 1(3):471–500, 2003.

[50] Michael L. Minion. Semi-implicit projection methods for incompressible flow based on spectral deferred corrections. *Appl. Numer. Math.*, 48(3-4):369–387, 2004. Workshop on Innovative Time Integrators for PDEs.

[51] B.D. Nichols and C.W. Hirt. Methods for calculating multidimensional, transient free surface flows past bodies. In *Proc. of the First International Conf. On Num. Ship Hydrodynamics*, pages 20–23, Gaithersburg, Oct. 1975.

[52] K. B. Nielsen. *Numerical Prediction of Green Water Loads on Ships*. PhD thesis, Maritime Engineering, MEK, Technical University of Denmark, 2003.

[53] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2003.

[54] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

[55] D. Pathria. The correct formulation of intermediate boundary conditions for Runge-Kutta time integration of initial-boundary value problems. *SIAM J. Sci. Comput.*, 18(5):1255–1266, 1997.

[56] Michael Pedersen. *Functional analysis in applied mathematics and engineering*. Studies in Advanced Mathematics. Chapman & Hall/CRC, Boca Raton, FL, 2000.

[57] Joseph Proriol. Sur une famille de polynomes à deux variables orthogonaux dans un triangle. *C. R. Acad. Sci. Paris*, 245:2459–2461, 1957.

[58] W. H. Reed and T. R. Hill. Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.

[59] Murray Rudman. Volume-tracking methods for interfacial flow calculations. *Internat. J. Numer. Methods Fluids*, 24(7):671–691, 1997.

[60] J. M. Sanz-Serna, J. G. Verwer, and W. H. Hundsdorfer. Convergence and order reduction of Runge-Kutta schemes applied to evolutionary problems in partial differential equations. *Numer. Math.*, 50(4):405–418, 1987.

[61] S. J. Sherwin and G. E. Karniadakis. A triangular spectral element method; applications to the incompressible Navier-Stokes equations. *Comput. Methods Appl. Mech. Engrg.*, 123(1-4):189–229, 1995.

[62] Chi-Wang Shu. Different formulations of the discontiniuous galerkin method for the viscous terms. Scientific Computing Report Series 2000 BrownSC-2000-07, Scientific Computing Group, Division of Applied Mathematics, Brown University, USA, 2000.

[63] M. Sussman, E. Fatemi, P. Smereka, and S. Osher. An improved level set method for incompressible two-phase flows. *Computers and Fluids*, 27(5-6):663–680, 1998.

[64] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two phase flow. *J. Comput. Phys.*, 114(1):149–159, 1994.

[65] Mark Sussman. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *J. Comput. Phys.*, 187(1):110–136, 2003.

[66] Mark Sussman and Emad Fatemi. An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow. *SIAM J. Sci. Comput.*, 20(4):1165–1191 (electronic), 1999.

[67] Mark Sussman and M. Y. Hussaini. A discontinuous spectral element method for the level set equation. *J. Sci. Comput.*, 19(1-3):479–500, 2003. Special issue in honor of the sixtieth birthday of Stanley Osher.

[68] P. G. Thomsen. A generalized runge kutta method of order three. Technical report, Informatics and Mathematical Modelling, IMM, Technical University of Denmark, 2002.

[69] Anna-Karin Tornberg and Björn Engquist. Regularization techniques for numerical approximation of PDEs with singularities. *J. Sci. Comput.*, 19(1-3):527–552, 2003. Special issue in honor of the sixtieth birthday of Stanley Osher.

[70] Lloyd N. Trefethen. *Spectral methods in MATLAB*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[71] O. Ubbink and R. I. Issa. A method for capturing sharp fluid interfaces on arbitrary meshes. *J. Comput. Phys.*, 153(1):26–50, 1999.

[72] T. Warburton, L. F. Pavarino, and J. S. Hesthaven. A pseudo-spectral scheme for the incompressible Navier-Stokes equations using unstructured nodal elements. *J. Comput. Phys.*, 164(1):1–21, 2000.

[73] T. C. Warburton, I. Lomtev, R. M. Kirby, and G. E. Karniadakis. A discontinuous Galerkin method for the compressible Navier-Stokes equations on hybrid grids. In *Tenth Int. Conf. on Finite Elements in Fluids*, January 5-8 1998.

[74] Tim Warburton. USEMe - nodal unstructured spectral element code. http://www.useme.org.

[75] Eric W. Weisstein. Vandermonde matrix. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/VandermondeMatrix.html.

[76] Steven T. Zalesak. Fully multidimensional flux-corrected transport algorithms for fluids. *J. Comput. Phys.*, 31(3):335–362, 1979.

[77] Yong-Tao Zhang and Chi-Wang Shu. High-order WENO schemes for Hamilton-Jacobi equations on triangular meshes. *SIAM J. Sci. Comput.*, 24(3):1005–1030 (electronic), 2002.