Towards Next Generation Ocean Models: Novel Discontinuous Galerkin Schemes for 2D unsteady biogeochemical models

by

Mattheus Percy Ueckermann

BASc., University of Waterloo (2007)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author .....
Department of Mechanical Engineering
September 2009

Certified by .....
Pierre F. J. Lermusiaux
Associate Professor of Mechanical Engineering
Thesis Supervisor

Accepted by .....
David E. Hardt
Chairman, Department Committee on Graduate Theses

Towards Next Generation Ocean Models: Novel Discontinuous Galerkin Schemes for 2D unsteady biogeochemical models

by

Mattheus Percy Ueckermann

Submitted to the Department of Mechanical Engineering
on September 2009, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

A new generation of efficient parallel, multi-scale, and interdisciplinary ocean models is required for better understanding and accurate predictions. The purpose of this thesis is to quantitatively identify promising numerical methods that are suitable to such predictions. In order to fulfill this purpose, current efforts towards creating new ocean models are reviewed, an understanding of the most promising methods used by other researchers is developed, the most promising existing methods are studied and applied to idealized cases, new methods are incubated and evaluated by solving test problems, and important numerical issues related to efficiency are examined.

The results of other research groups towards developing the second generation of ocean models are first reviewed. Next, the Discontinuous Galerkin (DG) method for solving advection-diffusion problems is described, including a discussion on schemes for solving higher order derivatives. The discrete formulation for advection-diffusion problems is detailed and implementation issues are discussed. The Hybrid Discontinuous Galerkin (HDG) Finite Element Method (FEM) is identified as a promising new numerical scheme for ocean simulations. For the first time, a DG FEM scheme is used to solve ocean biogeochemical advection-diffusion-reaction equations on a two-dimensional idealized domain, and p-adaptivity across constituents is examined. Each aspect of the numerical solution is examined separately, and p-adaptive strategies are explored. Finally, numerous solver-preconditioner combinations are benchmarked to identify an efficient solution method for inverting matrices, which is necessary for implicit time integration schemes. From our quantitative incubation of numerical schemes, a number of recommendations on the tools necessary to solve dynamical equations for multiscale ocean predictions are provided.

Thesis Supervisor: Pierre F. J. Lermusiaux
Title: Associate Professor of Mechanical Engineering

Acknowledgments

Many thanks to my thesis advisor, Pierre, for his guidance throughout the process of this thesis work. Particularly, I thank him for his careful reading of this document, and his suggestions for improvements. Also, thanks to research scientists Pat and Wayne, Pat for his patience and help with debugging, and Wayne for all his constructive criticisms. Thanks to Oleg and Jinshan for all their comments and suggestions.

Thanks to Themis for his many fun and enlightening discussions, Arpit for his attention to detail when checking my work, and Lisa for her friendliness, professionalism, and her support with the biogeochemical test cases. Thanks to Eric for his help with Latex, technical details, and his friendship. Thank you Melissa and Aprit for help proofreading my thesis. I also thank Mike for many fun discussions on numerics, and Sarah for listening to my every complaint. Also Harry, whenever I need a friend you are always there, thank you for that.

To mom and dad, thanks so much for always supporting me in whatever I do. Thanks for pushing when I need a push, and pulling when I am pushing too hard. Thanks to my sister Anabel who has always been an inspiration.

I am grateful to the Massachusetts Institute of Technology for awarding me the Pappalardo Fellowship, the Office of Naval Research for research support under grants N00014-07-1-1061 (ONR6.1) N00014-08-1-1097(PHILEX) to the Massachusetts Institute of Technology, and also the Natural Sciences and Engineering Research Council of Canada for awarding me a scholarship to aid with my financial support.

I am very grateful for this opportunity given to me, and I feel truly blessed for being surrounded by so many wonderful people.

Glossary

ε_h	The set of discretized edges
ε_h^∂	The set of discretized edges on the boundary of the domain Ω
ε_h^o	The set of discretized edges on the interior of the domain Ω
θ	A generic (modal or nodal) basis function
Ω	The domain of interest
$\partial\Omega$	The boundary of the domain of interest
ϕ	A nodal basis function
ψ	A modal basis function
$\mathbf{C}$	Convection (or stiffness) matrix $\mathbf{C}_{ji}^k = \int_K \theta_i(\mathbf{x}) \cdot \nabla\theta_j(\mathbf{x}) dK$
$\mathbf{F}$	The functional form of the flux
K	A single element in the triangulation
∂K	The boundary of a single element in the triangulation
$\mathbf{M}$	The mass-matrix, where $\mathbf{M}_{ji} = \int_\Omega \theta_i\theta_j d\Omega$
$\hat{\mathbf{n}}$	The unit normal vector pointing out of the domain
$\mathcal{P}^p$	Set of polynomials of order p
$\mathbf{q}$	The scaled gradient of u, that is, $\mathbf{q} - \kappa\nabla u = 0$
R	The residual
$S, \mathbf{S}$	The scalar and vector functional forms of the source term
$\mathcal{T}_h$	The discretized triangulation
$u, \mathbf{u}$	The unknown scalar or vector respectively
$\mathbf{v}$	Vector weighting (or test) function
$\mathcal{V}$	Generalized Vandermonde matrix
w	Scalar weighting (or test) function
$\mathbf{x}$	Spatial coordinates
ADR	Advection-Diffusion-Reaction
BiCGSTAB	Bi-Conjugate Gradient STABilized
CG	Continuous Galerkin
CDG	Compact Discontinuous Galerkin
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy: Numerical stability condition
CGS	Conjugate Gradient Squared
DG	Discontinuous Galerkin
DOF	Degree(s) of Freedom
FD	Finite Difference
FEM	Finite Element Method
FV	Finite Volume
GCM	General Circulation Model
GMRES	Generalized Minimum RESidual
GS	Gauss-Seidel
h-adaptive	Mesh adaptation strategy based on refining/coarsening elements
HDG	Hybrid Discontinuous Galerkin
HS	Hydrostatic
IBM	Immersed Boundary Method
ILU	Incomplete Lower Upper factorization
IP	Internal Penalty method
LDG	Local Discontinuous Galerkin

LU	Lower Upper factorization
MG	Multi-Grid
MPI	Message Passing Interface: A parallel programming language
MWR	Method of Weighted Residuals
NHS	Non-Hydrostatic
NPDZ	Nutrient-Phytoplankton-Detritus-Zooplankton: A four-component biological model
NPZ	Nutrient-Phytoplankton-Zooplankton: A three-component biological model
p-adaptive	Mesh adaptation strategy based in increasing/decreasing the polynomial order of basis functions
PE	Primitive Equations
QMR	Quasi-Minimum Residual
RK	Runge-Kutta: A time discretization scheme
RKDG	Runge-Kutta Discontinuous Galerkin
S-coordinates	Sigma coordinates: A terrain-following vertical discretization scheme
SSP	Strong Stability Preserving: Type of RK scheme
SWE	Shallow Water Equations
WHOI	Woods Hole Oceanographic Institute
Z-coordinates	A stair-case vertical discretization scheme

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The impact of human activities on the ocean and lakes is becoming increasingly global. To successfully coexist with the ocean and utilize marine resources, civilization needs to monitor and predict our natural environment. A new generation of efficient parallel, multi-scale, and interdisciplinary ocean models is required for better understanding and accurate predictions. There is a rich spectrum of needs for ocean modeling, including climate dynamics, the sustenance of life on Earth, coastal ocean and fisheries management, biological production and ecosystem dynamics, efficient maritime route planning, hazardous spills dispersion, and underwater sound propagation for efficient naval operations. Ocean prediction is a challenging problem due to its multi-disciplinary and multi-scale nature, and due to the constraint of real-time predictions. Depending on the phenomena being examined, space scales can vary from millimeters to planetary, and time scales can vary between seconds to millenniums. Also, for accurate simulation results, efficient nonlinear assimilation of data into ocean models and estimation of the most useful data using adaptive sampling is required.

The MIT "Multidisciplinary Simulation, Estimation and Assimilation System" (MSEAS) (Web-MSEAS, 2009), includes the primitive equation code of the Harvard Ocean Prediction System (HOPS) and other computational systems: a nested data-assimilative barotropic tidal prediction system (Logutov and Lemusiaux, 2008), a coastal objective analysis scheme, the Error Subspace Statistical Estimation (ESSE)

system for data assimilation (Lemusiaux, 1999), optimization (Heaney et al., 2007) and adaptive sampling (Lemusiaux, 2007), novel Objective Analysis schemes (Agarwal, 2009), multiple biological models (Besiktepe et al., 2003) and several acoustic models (Robinson and Lermusiaux, 2003). This system is being used for realistic simulations and real-time forecasts in many regions of the world's ocean. At the heart of this system is a free-surface hydrostatic primitive equation model with new two-way nesting capabilities. These capabilities have been used in real-time experiments since 2001 to improve the resolution accuracy in selected regions with minimal modification and run-time expense.

One of the goals of the MSEAS group is to utilize and develop new numerical methods for ocean predictions. In the past decade, new numerical algorithms have been developed, not only for computational fluid dynamics, but also for chemical and biological dynamics. It is now possible to research the next generation of ocean prediction models that build upon progress made in these other research fields, leading to a better understanding of interdisciplinary ocean dynamics. Ocean specific numerical research includes: fully coupled physical, biological and acoustic modeling; multi-scale models; unstructured spatial grids; distributed ocean modeling; embedded models; high-order schemes; as well as self-modifying models that adapt to data and learn proper parameterizations and parameters.

The purpose of this thesis is to identify promising numerical methods that are suitable to ocean predictions. In order to fulfill this purpose, current efforts towards creating new ocean models are reviewed, an understanding of the most promising methods used by other researchers is developed, new methods are investigated and demonstrated by solving a test problem, and important numerical issues related to efficiency are examined. The Discontinuous Galerkin (DG) Finite Element Method (FEM) is identified as a promising new numerical scheme for ocean simulations. The DG FEM is used to solve biogeochemical advection-diffusion-reaction equations on a two-dimensional idealized domain, and p-adaptivity across constituents is examined. Finally, the efficient inversion of the linear discrete operator using iterative solvers is explored. This thesis develops the tools necessary to solve the dynamical equations

for ocean predictions.

1.1 Thesis Organization

Chapter 2 reviews the work done by numerous groups towards developing the second generation of ocean models. The model developed by each group is briefly summarized, and all the models are compared and grouped. Chapter 3 describes the Discontinuous Galerkin (DG) method for solving advection-diffusion problems, detailing the discrete formulation and discussing implementation issues. The new Hybrid Discontinuous Galerkin method for solving higher order derivatives is also briefly discussed. Chapter 4 demonstrates the solution of biogeochemical reaction equations on two-dimensional unstructured grids using DG. Each aspect of the numerical solution is examined separately, and p-adaptive strategies are examined. Chapter 5 benchmarks numerous solver-preconditioner combinations to identify an efficient solution method for inverting matrices, which is necessary for implicit time integration schemes. Finally, Chapter 6 summarizes the conclusions and makes recommendations on how to proceed.

Chapter 2

Review of Ocean Models

2.1 Introduction

The first generation of ocean modelling systems are based on the seminal article by Bryan (1969). In this article a hydrostatic, rigid lid model is proposed with an energy conserving numerical scheme. While modern ocean models have become sophisticated modelling systems with complex data assimilation schemes, adaptive modelling capabilities, free surface [and] open boundary conditions, the numerical schemes used for these models are still largely based on the original computational fluid dynamics technology of the late sixties, that is low-order finite difference and finite volume schemes on structured grids. For a review of the first generation of ocean models, the reader is referred to Griffies et al. (2000).

Recent advances in numerical schemes include finite volume and finite elements methods on unstructured grids. While some ocean models have used the finite volume methods (Marshall et al., 1997a,b, 1998), all of the first generation modelling systems are based on low order schemes on structured grids. The vertical discretization has garnered significant attention, resulting in a number of terrain following coordinate schemes (Freeman et al., 1972), isopycnal vertical coordinates (Bleck and Smith, 1990), z-coordinates (Bryan, 1969), and hybrid schemes (Spall and Robinson, 1990, Pietrzak et al., 2002). Also, curvilinear structured grids have been used in the horizontal (Adcroft et al., 2004). Nonetheless, it has been recognized by a number

of different modelling groups that new Computational Fluid Dynamics (CFD) technologies are suitable to be used for the second generation of ocean models. The most prominent second generation models are summarized in Table 2.1, where "second generation" is [for now] interpreted as those models that use unstructured grids. Refer to Table A.1 for an additional summary with more details.

In the following sections, each of these models are described individually to highlight the different modelling ideologies, features, and numerical methods. For a general review of modelling efforts, the reader is referred to Pain et al. (2005) and Slingo et al. (2009).

2.2 ADCIRC

ADCIRC is a FEM model developed for coastal oceans, shelves, estuaries, inlets, floodplains, rivers and beaches. The development team consists of R. Luettich (UNC-CH), J. Westerink (ND) R. Kolar (OU), C. Dawson (UT), S. Bunya (U-Tokyo), and E. Kubatko (OSU).

The model is actively being developed with current efforts towards upgrading the computational engine from a CG FEM based solution to a new h-p adaptive DG FEM based algorithm. The model can solve the following equations: two-dimensional Shallow Water Equations (SWE); three-dimensional mass and momentum conservation subject to incompressibility, hydrostatic and Boussinesq approximations; two-dimensional sediment continuity equation; two-dimensional and three-dimensional temperature and salinity transport equations.

Some **features** of the model include:

- full wetting/drying elements in two and three dimensions;
- barrier elements (such as levees);
- Conduits and porous barriers; at least second order accurate numerical schemes;
- implicit or explicit time-stepping schemes;
- highly scalable parallel Message Passing Interface (MPI) implementation (up to 1000's of processors). This system is written in FORTRAN 90.

Model Name	Details
ADCIRC *ADvanced CIRCulation model*	FEM (CG or DG). Designed for coastal oceans, shelves, estuaries, inlets, floodplains, rivers and beaches
Delfin	FV/FD
ELCIRC *Eularian-Lagrangian CIRCulation model*	FV/FD Eulerian-Lagrangian using prisms/quads. Developed for Columbia River, also used for simulation of 3D baroclinic circulation across river-to-ocean scales, and for estuaries and continental shelves.
FEOM *Finite Element Ocean Model*	FEM using prisms. General-purpose general circulation model solving primitive equations under Boussinesq approximation
Finel	FEM using tetrahedrals. Solves 3D non-hydrostatic equations.
FVCOM *Finite Volume Coastal Ocean Model*	FV using prisms. Developed for estuarine flooding/drying process in estuaries and the tidal-,buoyancy- and wind-driven circulation in coastal regions featured with complex irregular geometry and steep bottom topography.
ICOM *Imperial College Ocean Model*	FEM (CG and DG) using tetrahedrals. Developed as general model useful for all ocean regimes.
RiCOM *River and Coastal Ocean Model*	FEM. Used to provide storm surge forecasts. Emphasis on coastal oceans.
SELFE *Semi-Eularian-Lagrangian Finite Element ocean model*	FEM using prisms. Developed for Columbia River, also used for simulation of 3D baroclinic circulation across river-to-ocean scales.
SEOM *Spectral Element Ocean Model*	Spectral Methods (SM). Solved 2D SWE, and solving primitive 3D Boussinesq equations in development.
SLIM *Second-generation Louvain-la-Neuve Ice-ocean Model*	FEM (DG or CG). Focus on global climate evolution.
SUNTANS *Standford Unstructured Non-hydrostatic Terrain-following Adaptive Navier-Stokes Simulator*	FV using prisms. Developed for coastal ocean simulations.
UnTRIM *Unstructured Tidal Residual Inter-tidal Mudflat model*	FV/FD using prisms/quadrilaterals. Developed for rivers, lakes, and coastal oceans.

Table 2.1: Summary of second generation ocean models

For recent articles related to the development of ADCIRC, refer to Dawson and Proft (2002), Bunya et al. (2005), Kubatko et al. (2006), and Forbes et al. (2007). Also the development site is located at Web-ADCIRC (2006).

This modelling system has been used for a number of applications including modelling tides (Westerink et al., 1994, Blanton et al., 2004, Jarosz et al., 2005), hurricane storm surges (Blain et al., Gica et al., 2001), flooding (Luettich and Westerink, 1995, Feyen et al., 2006), and wind driven circulation and transport (Luettich et al., 1999). It has also been used extensively for storm surge simulations in New Orleans (Westerink et al., 2007). Finally, it is used by the U.S. Army Corps of Engineers and the U.S. Navy, is certified by FEMA for the National Flood Insurance Program, and is used by NOAA's National Ocean Services for storm surge/inundation applications.

2.3 Delfin and Finel

Delfin was developed by D. Ham under the supervision of J. Pietrzak and Guus Stelling at Delft University of Technology (Ham, 2006). This model is a three-dimensional finite-volume/finite-difference model using an unstructured mesh. The group for which this model was developed is currently studying the Indian Ocean Tsunami. D. Ham is currently working with the ICOM group. This code is written in C.

Finel was also developed by the same group, and it is a three-dimensional non-hydrostatic finite element model bases on a tetrahedral mesh (that is, unstructured in all three dimensions). This code is being developed by R. J. Labeur at TU Delft. The website group website is located at Web-DELFT (2009).

2.4 ELCIRC and SELFE

ELCIRC and SELFE were originally developed for applications surrounding the Columbia river estuary. The CORIE modeling system, a coastal margin observatory for the Columbia River estuary and plume, uses SELFE and ELCIRC by default, but

have previously used other models such as POM, ADCIRC, and QUODDY. SELFE is the newest model and has a number of improvements over ELCIRC, overcoming restrictions in the discretization. SELFE uses the FEM, while ELCIRC uses FV in the horizontal and FD in the vertical. The group is based out of the OGI School of Science and Engineering, and the development team consists of A. Baptista (Scientific director), J. Zhang, M. G. G. Foreman, D. Stucchi, E.P. Myers, A. Oliveira, and A.B. Fortunato. The model is actively being developed, with current efforts towards solving non-hydrostatic equations.

The current model solves the 3D shallow-water equations, with hydrostatic and Boussinesq approximations, and transport equations for salinity and heat. The primary variables that SELFE solves for are: the free-surface elevation; 3D velocity; 3D salinity; and 3D temperature of the water. The numerical formulation is not explicitly mass-conserving, but the mass-conservation properties are "very good." Neither ELCIRC nor SELFE use a mode-splitting scheme, nor do they use a projection method, that is, the velocity and surface elevation are solved simultaneously (Zhang et al., 2004, Baptista and Zhang, 2008). SELFE and ELCIRC use quadrilateral or prismatic elements, allowing great flexibility in the choice for vertical discretization.

Some **features** of the model include:

- wetting and drying;
- Z, S, or mixed S-Z coordinates for vertical discretization;
- ECO-SELFE (biological model);
- Semi-implicit time integration;
- Both parallel (MPI) and serial version of code.

This system is written using a combination of FORTRAN and MATLAB.

A recent description of SELFE can be found in Baptista and Zhang (2008), and a description of ELCIRC can be found in Zhang et al. (2004). Also, the group website is located at Web-CORRIE (2009).

ELCIRC has been applied to the Columbia River (Baptista et al., 2005), to the St. John's river (Myers and Aikman, 2003), to study marine ecosystem connectivity

(Robinson et al., 2005), and to stratification (Pinto et al., 2003) and tidal (Foreman et al., 2006) studies in estuaries. SELFE has been tested extensively against standard ocean/coastal benchmarks and used in a number of bays/estuaries around the world (Baptista and Zhang, 2008). SELFE will be used for the same applications as ELCIRC in the future.

2.5 FEOM

FEOM is being developed under the Community Ocean Model (COM) project undertaken by the Alfred Wegener Institute (AWI) located in Bremerhaven, Germany. The goal of the COM project is to develop a general-purpose ocean model based on unstructured meshes that contains standard ocean modelling tools, such as different advection schemes, mixed layer parameterizations, free surface boundary conditions, and generalized vertical coordinates. FEOM uses the FEM. This is an open source project, with a number of approved developers, but the main contacts are S. Danilov, L. Nerger, and J. Schröter.

This model is actively being developed with an emphasis on making this unstructured grid model as efficient as a structured grid model. FEOM solves the 3D primitive equations under the Boussinesq approximation. It uses prismatic elements, allowing for a generalized vertical discretization. An earlier, less-efficient version of the code used prismatic elements.

Some of the model **features** include:

- Free surface;
- Non-hydrostatic;
- Sea-ice model;
- NPDZ biological model;
- Semi-implicit time stepping;
- Parallel MPI implementation.

The formulation of FEOM is described in Danilov et al. (2004), and effects of vertical discretization are discussed in Wang et al. (2008). The AWI website is located at Web-AWI (2009) and the FEOM project page is located at Web-FEOM (2009).

The model has been used for studying circulation and bottom pressure in the Atlantic (Böning et al., 2006), assimilation of sea-surface height data from the TANDAM project (Nerger et al., 2006), and studying the influence of tidal forcing and topography representation in the Weddel Sea (Wang et al., 2009).

2.6 FVCOM

FVCOM was originally developed for the estuarine wetting/drying process in estuaries and the tidal-, buoyancy- and wind-driven circulation in coastal regions with complex irregular geometry and steep bottom topography. FVCOM uses the FV method. The FVCOM group is based at the University of Massachusetts-Dartmouth, and the main development team consists of C. Chen (UMass), G. Cowles (UMass), and R. C. Beardsley (WHOI).

This model is mature, but is still actively being developed, with current focus on the non-hydrostatic solver. The hydrostatic model solves the 3D primitive equations with a mode-splitting scheme. This model uses prismatic elements, with the vertical discretization employing terrain-following coordinates. FVCOM is a complete modelling system, with some of the model **features** including:

- Free surface;
- Non-hydrostatic;
- wetting/drying elements;
- Biological models;
- Fully non-linear ice models;
- Wave model;
- Semi-implicit time stepping;
- Parallel MPI implementation.

The system is written in FORTRAN 90.

The formulations for FVCOM is described in Chen et al. (2003), with a comparison between structured and unstructured grids in Chen et al. (2007). The group website is located at Web-FVCOM (2009).

FVCOM has seen a number of applications in Bays, (Zhao et al., 2006, Chen et al., 2008), estuaries (Xue et al., 2009), lakes (Chen et al., 2003), seas (Chen et al., 2003) and other regimes. A complete listing of current projects can be found at Web-FVCOM (2009).

2.7 ICOM

ICOM is being developed for use as a general ocean circulation model. The model uses sophisticated anisotropic mesh adaptivity in three dimensions. The ICOM group is based out of the Imperial College in London, but collaborates with Oxford, the National Oceanography Center in Southampton, and the Proudman Oceanographic Laboratory in Liverpool. ICOM uses either the CG FEM or the DG FEM, with research toward determining the best type of finite element to use for ocean applications. Some of the developers responsible for developing the model are C. C. Pain (Project head), D. A. Ham, M. D. Piggot, C. J. Cotter, A. J. H Goddard, C. R. E. De Oliveira, and A. P. Umpleby.

A large team is actively developing this model. ICOM uses tetrahedral elements with anisotropic adaptivity in all three dimensions. The model solves the three-dimensional non-hydrostatic Boussinesq equations. A projection method is used to enforce the continuity constraint. Some of the model **features** include:

- Sophisticated dynamic anisotropic mesh adaptivity;
- Free-surface;
- Non-hydrostatic;
- NPDZ biology model;
- wetting and drying;
- Implicit time stepping;

- Sophisticated load-balanced domain decomposition parallel implementation.

The model is described in Ford et al. (2004a) and Piggott et al. (2008), and validated in Ford et al. (2004b). Mesh adaptivity is discussed in Piggott et al. (2005) with the optimization metric discussed in Power et al. (2006). The development website is located at Web-ICOM (2008).

This model is still in the development phase and has not seen much realistic application. It promises to be useful for modelling Western boundary currents, flow over topography, open ocean deep convection, gravity currents, internal wave breaking, salt fingering, tidal modelling, tsunami modelling, North Atlantic thermohaline circulation, and wetting and drying.

2.8 RiCOM

RiCOM was developed by R. A. Walters, and is used to provide storm surge forecasts for New Zealand. It solves the three-dimensional primitive equation hydrodynamic model with semi-implicit time stepping and a semi-Lagrangian advection scheme. It uses the CG FEM on triangular and quadrilateral elements. It also has a non-hydrostatic pressure option. This model is embedded into a New Zealand forecasting system that includes a Local Area Weather model, a sea surface height model, and a wave model.

The model formulation is described in Walters (2005b) and validation for storm surge forecasting is discussed in Lane and Walters (2009). The RiCOM model has results related to ocean modelling, from the type of FEM to use (Walters, 2005a, Walters and Barragy, 1997), to solution methods (Barragy and Walters, 1998, Walters et al., 2007) to model design considerations (Walters, 2006).

2.9 SEOM

SEOM is being developed for large scale ocean applications. The eventual goal is to solve the three-dimensional non-hydrostatic primitive equations, but SEOM currently has a robust solver for the two-dimensional, depth integrated shallow water equations. SEOM3D solves the primitive hydrostatic and Boussinesq Navier Stokes equations in three dimensions. This model uses high-order Spectral Element Method (SEM) on unstructured rectangular meshes in the horizontal and sigma coordinates in the vertical. SEOM is developed by M. Iskandarani (Project head), D. B. Haidvogel, J. C. Levin, and J. P. Boyd.

Some of the model **features** include:

- *h-p* refinement;
- Free-surface;
- Non-hydrostatic;
- Semi-implicit time stepping;
- MPI parallel implementation.

SEOM was originally written in C, but was later re-coded in FORTRAN 90.

The formulation of the two-dimensional SEOM is described in Iskandarani et al. (1995), and the three-dimensional formulation is described in Iskandarani et al. (2003). The SEOM development website is located at: Web-SEOM (2009).

The two-dimensional version of SEOM has seen a number of applications, including an investigation of the wind-driven circulation in the Mediterranean sea (Molcard et al., 2002) and an investigation of the dynamics of the long period tides in the global ocean (Wunsch et al., 1997).

2.10 SLIM

SLIM is developed as an ice-ocean model for simulating global climate, and large-scale ocean applications. SLIM is based out of the UCL, with partners from all

around the world. SLIM uses both the CG FEM and the DG FEM with research aimed towards determining the best type of finite element for ocean applications. SLIM is being developed by a large team, some members including E. Deleersnijder, T. Fichefet, V. Legat, J.-F. Remacle, E. Hanert, C. König Beatty, L. White (UCL), J.-M. Beckers (ULG), V. Dehant (Royal Observatory of Belgium), O. de Viron (IPGP), E. Delhez (ULG), E. Hanert from the (UoR, UK), D. Le Roux (ULaval), and E. Wolanski (AIMS).

The model is actively being developed with current efforts focused on the three-dimensional implementation. SLIM uses horizontally adaptive unstructured prismatic meshes. The governing equations currently solved for include the depth-integrated shallow-water equations and the three-dimensional hydrostatic primitive equations. The SLIM group has generated a number of high-quality unstructured meshes for various ocean regimes. Some of the current model **features** include:

- Adaptive horizontal mesh;
- Sea-ice model;
- State of art sub-grid-scale parameterizations;
- Semi-implicit time stepping;
- Parallel.

The SLIM project has computational results in tracer advection (White, 2008), mesh adaptation (Remacle et al., 2005, 2006), mesh generation (Legrand et al., 2007), and dispersion analysis (Bernard et al., 2008, Le Roux, 2005). The two-dimensional formulation is outlined in Le Roux et al. (2000) and the three-dimensional formulation is outlined in White et al. (2008). The group development website is located at Web-SLIM (2009).

Since the model is still under development, it has not seen a large number of realistic applications. Once completed, it should be useful as a general ocean circulation model with application to multi-scale ocean processes.

2.11 SUNTANS

SUNTANS is an unstructured grid, non-hydrostatic, finite volume coastal ocean simulator. The development team is based out of Stanford, but includes a number of researchers from around the world. The core development team at Stanford consists of A. Boehm, D. Fong, O. Fringer, M. Gerritsen, E. Gross, J. Koseff, S. Monismith, R. Naylor, R. Street, and S. Sankaranarayanan. The model is mature, and one of the current development projects is nesting this unstructured grid model within the existing structured grid Rutgers Regional Ocean Model System (ROMS) (Fringer et al., 2006a). SUNTANS uses prismatic meshes and a stair-case representation of the bottom topography. The stair-case representation is combined with the Immerse Boundary Method (IBM) to properly resolve bottom topography. The model solves the Navier Stokes equations under the Boussinesq assumption, and uses an LES turbulence closure model. Some of the model **features** include:

- Non-hydrostatic;
- Large eddy simulation for resolved features;
- Z-level coordinates combined with the immersed boundary method for accurate topography on bottom;
- wetting and drying;
- Semi-implicit time stepping using the Theta method;
- Parallel (MPI) implementation.

The formulation for SUNTANS is described in Fringer et al. (2006b), and the model is based on the method by Casulli (1999). The group development site is located at Web-SUNTANS (2009).

The majority of SUNTANS applications have been related to internal tides (Jachec et al., 2006, 2007, Venayagamoorthy and Fringer, 2005), but the code could also be used for applications in estuaries and coastal oceans.

2.12 UnTRIM

UnTRIM is an unstructured orthogonal grid finite volume or finite difference model using prismatic or quadrilateral elements. It was developed for estuaries, lakes, and coastal oceans. It was developed by V. Casulli from Trento University, Italy. The model solves the three-dimensional shallow water equations, as well as three-dimensional transport equations for salt, heat, dissolved matter, and suspended sediments. Some of the model **features** include:

- Non-hydrostatic;
- Free surface;
- Semi-implicit time stepping;
- Parallel (MPI) implementation.

Publications related to the development of UnTRIM are: Casulli (1999), Casulli and Walters (2000), Casulli and Zanolli (2002, 2005), and the code can be obtained from: Web-UNTRIM (2009).

UnTRIM has a large user base, and has been applied to storm surge predictions in bays (Shen et al., 2006a,b), and rivers (Liu et al., 2008).

2.13 Discussion and Conclusions

In order to compare and contrast the different models, Figure 2-1 was created. There are two main groups of models, the FEM and FV models. In general, the FV models are more mature, likely because the technology is older. In the FV group, FVCOM and SUNTANS have greater flexibility in their meshes, since UnTRIM requires orthogonal unstructured meshes. Also, FVCOM is a mature modelling *system*, not just a solver for dynamics. UnTRIM also has a large user base. But SUNTANS and FVCOM are expected to see increased usage in the future.

In general, the FEM models are less mature, highlighted by the fact that most do not have non-hydrostatic solver options. The ADCIRC, SELFE/ELCIRC, and

FEOM models have been used for realistic, albeit somewhat specialized, applications. RiCOM and DELFIN/FINLAB have fewer developers, and the models are less sophisticated than the top row. SEOM is still under development, although it has seen more realistic usage than either ICOM or SLIM. ICOM and SLIM are the most sophisticated models reviewed, with their major advantage being adaptive meshing, although this does lead to an increase in development time due to the additional complexity.

Most of the second generation models reviewed use the FEM with some form of non-conforming or discontinuous element. The FEM offers a number of advantages over the FV method. Specifically, the FEM variational framework allows closed from proofs about the numerical schemes in terms of consistency and stability. Also, higher order schemes are more easily formulated, providing a flexible code capable of arbitrarily high order schemes. The FEM can also be generalized for arbitrarily shaped elements, allowing a single implementation capable of having mixed elements within the same mesh. The FEM is more general than the FV method allowing greater flexibility when developing new schemes. In fact, FV methods can be cast in terms of the FEM. Among the disadvantages of traditional FEMs are increased complexity in the implementation, and CG FEMs have difficulty stabilizing advection-dominated flows, leading to complicated stabilization schemes. Newer DG schemes do not have difficulty stabilizing advection-dominated flows, but they suffer from poorer computational efficiency and complications with second order or higher derivatives. Despite the disadvantages, most of the second generation model developers chose to use the FEM with a non-conforming or discontinuous discretization.

Because the DG schemes are newer with less established practices, and because they offer exciting new possibilities for solving advection-dominated flows, it was decided to investigate these schemes further. The next section provides and overview of DG methods.

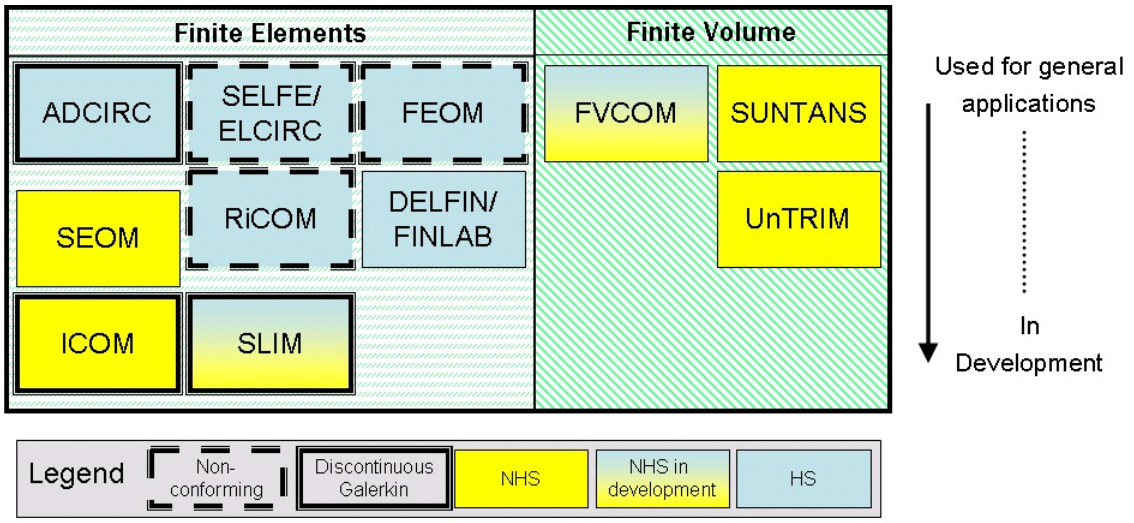

Figure 2-1: Second generation unstructured grid ocean modelling systems

Chapter 3

Discontinuous Galerkin (DG) Methods

It is assumed that the reader is familiar with FD and FV methods, but that a brief review of the FEM is in order.

First a consistent notation is introduced that will be used throughout this document. Referring to Figure 3-1, the problem domain is specified as Ω, and its boundary as $\partial\Omega$. If a boundary has a specified type "D", that boundary will be indicated as $\partial\Omega_D$. The discretized triangulation is represented by $\mathcal{T}_h$. Individual elements within $\mathcal{T}_h$ are represented with K_i, where the subscript is used to refer to a specific triangle in the triangulation or omitted when referring a general triangle. The boundary of an element K_i is indicated by ∂K_i. Thus we can say $\mathcal{T}_h = \bigcup K_i$. We also define the set containing all the edges in the domain $\varepsilon_h = \bigcup \partial K_i$, the set containing all domain-boundary edges $\varepsilon_h^\partial = \varepsilon_h \cap \partial\Omega$, and the set containing all domain-interior edges $\varepsilon_h^\circ = \varepsilon_h \backslash \varepsilon_h^\partial$.

Consider:

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{F}(u) - S(u) = 0 \tag{3.1}$$

The numerical solution, u_h, to this equation will have a residual $R = \frac{\partial u_h}{\partial t} + \nabla \cdot \mathbf{F}(u_h) - S(u_h)$. In FEM, we try to set $R = 0$ over specified weighting (or test)

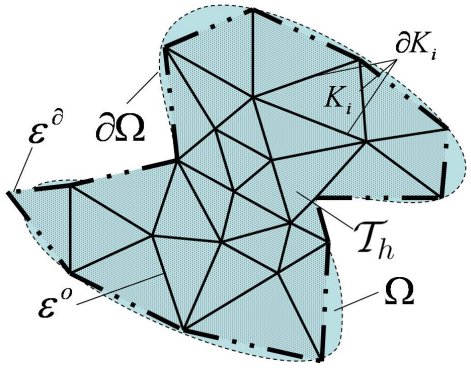

Figure 3-1: Notation definition for domain

functions on the element, and this is known as the Method of Weighted Residuals (MWR) (Chapra and Canale, 2006). That is, we set

$$\int_{\Omega} Rwd\Omega = 0 \tag{3.2}$$

where w is the test function. If w and the numerical solution u_h were in an infinite dimensional space, then u_h would satisfy the equations exactly. However, by the very nature of discrete, numerical solutions, the space of w and u_h cannot be infinite. Their numerical solutions but reside in a *finite* space. The choice of space will make a significant difference in the type of FEM and the solution method.

There are a number of standard choices for choosing the test function, including:

1. Collocation
2. Subdomain
3. Galerkin

With the Collocation method, $w = \delta(\mathbf{x}_i)$, that is, the test functions are chosen as delta functions at discretely chosen points $\mathbf{x}_i$. With the Subdomain method, $w = C|_K$, that is w is chosen as a constant, C, over the triangle K. With the Galerkin method, w is chosen to be the same as the basis function, θ, used to represent u_h, that is $w = \theta$.

To discretize the solution in FEM, a finite dimensional basis function that attempts to represent the shape of the true solution is used. This basis is finite and incomplete,

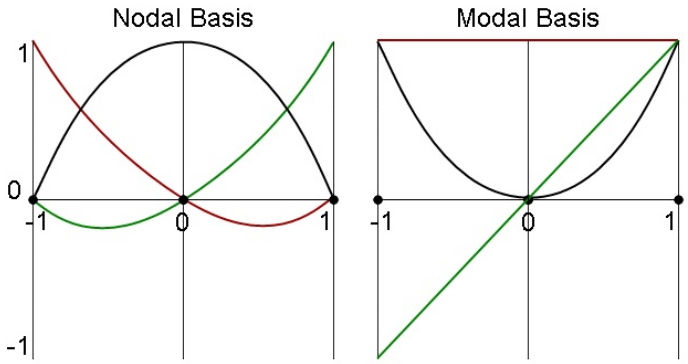

Figure 3-2: Example of one-dimensional quadratic nodal and modal bases

that is, it may not reproduce the true solution. Thus, we say that the continuous true function is expressed, for example, as

$$u(\mathbf{x},t) \approx u_h(\mathbf{x},t) = \sum_i u_{h,i}(t)\theta_i(\mathbf{x}) \tag{3.3}$$

$$u_h(\mathbf{x},t) = \sum_{i=1}^{N_P} u_{h,i}(t)\psi_i(\mathbf{x}) \tag{3.4}$$

$$u_h(\mathbf{x},t) = \sum_{i=1}^{N_P} u_{h,i}(t,\mathbf{x}_i)\phi_i(\mathbf{x}) \tag{3.5}$$

where 3.3 is a generic representation of a basis $\theta_i(\mathbf{x})$, 3.4 is an example of a modal basis function $\psi_i(\mathbf{x})$ where the unknown coefficients $u_{h,i}(t)$ are a function of time and related to a specific mode, and 3.5 is an example of a nodal basis function $\phi_i(\mathbf{x})$ where the unknown coefficients $u_{h,i}(t,\mathbf{x_i})$ are a function of time and related to a specific point in space $\mathbf{x}_i$, and N_p is the number of points or modes. Note that the notation "$(.)_h$" is used to indicate the discretized solution which is dependent on the mesh size characterized by the value "h". A nodal basis is equal to one at a particular node, and zero on all other nodes. A modal basis is usually non-zero on the entire element, but is related to a specific mode or polynomial power. An example of a one-dimensional quadratic nodal and model element is shown in Figure 3-2.

As an example, with this machinery in place, the discretization of $\int_\Omega \frac{\partial u}{\partial t} w d\Omega$ pro-

ceeds using 3.3 as follows:

$$\begin{aligned}\int_\Omega \frac{\partial u}{\partial t} w d\Omega &\approx \int_\Omega \frac{\partial \sum_i u_{h,i}\theta_i}{\partial t} w_h d\Omega \\ &= \sum_i \frac{\partial u_{h,i}}{\partial t} \int_\Omega \theta_i w_h d\Omega\end{aligned}$$

Choosing w_h in the Galerkin sense, that is $w_h = \theta_j . j = 1...N_P$, we have

$$\begin{aligned}\int_\Omega \frac{\partial u}{\partial t} w d\Omega &\approx \sum_i \frac{\partial u_{h,i}}{\partial t} \int_\Omega \theta_i \theta_j d\Omega \\ &= \sum_i \frac{\partial u_{h,i}}{\partial t} \int_\Omega \theta_i \theta_j d\Omega\end{aligned}$$

which can be written as a matrix-vector multiplication

$$\int_\Omega \frac{\partial u}{\partial t} w d\Omega \approx \mathbf{M} \frac{\partial \mathbf{u}_h}{\partial t}$$

where $\mathbf{M}_{ji} = \int_\Omega \theta_i \theta_j d\Omega$ is known as the mass matrix. Note, in this example, θ could be either a modal or nodal basis, and can be defined in any appropriate space. The FEM is a powerful numerical method that allows flexibility through the choice of basis and test functions. In particular, FD and FV schemes can be recovered using the FEM. The FEM also allows for great geometric flexibility since the formulation is not dependent on the discretization, enabling the use of unstructured grids.

3.1 Introduction to DG

The first reported use of DG FEM was by Reed and Hill (1973) where DG was used to solve the steady-state neutron transport equation. However, DG drew little attention until a series of papers (Cockburn and Shu, 1989, Cockburn et al., 1989, 1990, Cockburn and Shu, 1998b), where the Runge-Kutta DG methods were described. The extension of DG to higher order derivatives (Bassi and Rebay, 1997) made the method applicable to solving advection-diffusion equations, which can be extended to solving the Navier Stokes equations. Since the late 90's, DG has seen a number of

realistic applications in aerospace, solid mechanics, and electromagnetism to name a few.

The major theoretical difference between CG and DG lies in the approximation subspaces used. DG uses bases that are in normed space $L^2(\Omega)$ while CG uses bases that are in the Hilbert space $H^1(\Omega)$, that is, the function has to be continuous across elements. For a function $f(\mathbf{x})$ to be in $L^2(\Omega)$, it has to satisfy $\int_\Omega f(\mathbf{x})^2 d\Omega < \infty$, whereas a function in $H^1(\Omega)$ has to belong to a smaller space satisfying $\int_\Omega f(\mathbf{x})^2 + \nabla f(\mathbf{x}) \cdot \nabla f(\mathbf{x}) d\Omega < \infty$. Figure 3-3 illustrates the difference between a one-dimensional DG space, and a one-dimensional CG space (both using a nodal basis). Notice for the DG scheme the slope is undefined across the element boundary, and thus the solution cannot reside in $H^1(\Omega)$. Also note that in the example shown, the CG scheme has four degrees of freedom while the DG scheme has six degrees of freedom due to the doubling of information at element boundaries.

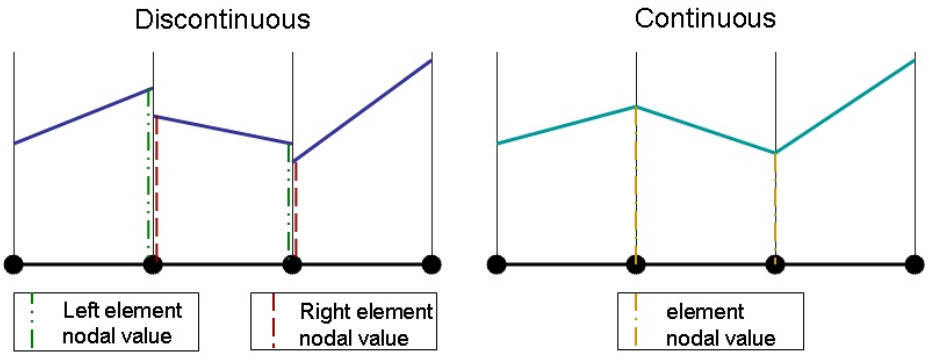

Figure 3-3: Difference between solution when using a discontinuous (left) or a continuous (right) basis

The duplication of unknowns is commonly quoted as a disadvantage of DG compared to CG, because there is an inherent increase in computational cost associated with a larger number of unknowns. However, proper studies comparing the error level (Kubatko et al., 2009) suggest that this disadvantage may not be as dramatic as stated for specific types of problems. The disadvantage of DG over FD methods is increased development time as well as decreased computational efficiency per degree of freedom. Apart from the efficiency issues, DG has a number of advantageous properties that promote its use, including:

- *Localized memory access patterns.* The local nature of DG elements allows improved scalability for parallel architectures, and promise to take better advantage of newer Graphics processing units (GPUs) that are geared towards massively parallel computations.

- *Higher order accuracy.* Since DG belongs to the FEM variational framework, the same interpolation theory applies. That is $\mathcal{O}(h^{p+1})$ convergence, where p is the order of the basis function used, can be obtained. Obtaining higher-order rates of convergence for FV on unstructured grids is difficult, and requires information from neighboring elements. Both FD and FV require large, non-compact stencils. The advantage for DG, then, is obtaining high-order convergence while maintaining the compact stencil.

- *Adaptive strategies.* The local nature of DG elements allows for a local element interpolation function of arbitrary order with no restrictions imposed by neighboring volumes. That is, the DG framework easily allows for non-conforming discretization which facilitates the use of h (adapting the triangulation) and p (adapting the order of the basis) adaptation strategies.

- *Designed for advection-dominated flows.* Where FV schemes struggle to achieve higher-order accuracy for advection, DG along with an appropriate Riemann solver easily generalizes to use arbitrarily high-order advection schemes for smooth solutions.

- *Superconvergence properties for dispersion and dissipation.* DG demonstrates superconvergence for the dispersion and dissipation of waves (Bernard et al., 2008), making it well suited to wave propagation problems.

- *Complex geometries.* Because DG fits into the FEM framework, it is easily generalized for use on arbitrarily shaped elements, making it suitable for use with unstructured grids to model complex geometries.

By using DG, then, one gains a great deal of flexibility in terms of flux stabilization schemes, geometry, and the order of the scheme at the cost of arguably greater

computational expense compared to CG, FV, and FD.

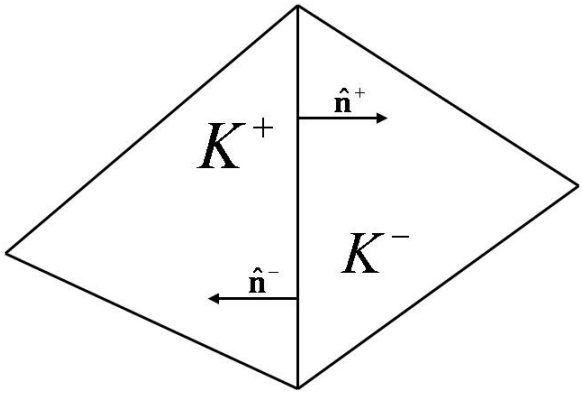

Figure 3-4: Notation for plus and minus triangular elements

Finally, some convenient notation to mathematically express the jumps across elements needs to be introduced. Often, in the literature, two elements bordering an edge are labeled K^+ and K^-, with associated outward pointing normals $\hat{\mathbf{n}}^+$ and $\hat{\mathbf{n}}^-$ respectively, as shown in Figure 3-4. (Alternatively, sometimes one element is referred to as the "left" while the other is referred to as the "right"). The mean values $\{\{.\}\}$ and jumps $[\![.]\!]$ are then defined as follows

$$\{\{\mathbf{v}\}\} = (\mathbf{v}^+ + \mathbf{v}^-)/2 \qquad \{\{w\}\} = (w^+ + w^-)/2$$
$$[\![\mathbf{v} \cdot \hat{\mathbf{n}}]\!] = \mathbf{v}^+ \cdot \hat{\mathbf{n}}^+ + \mathbf{v}^- \cdot \hat{\mathbf{n}}^- \qquad [\![w\hat{\mathbf{n}}]\!] = w^+\hat{\mathbf{n}}^+ + w^-\hat{\mathbf{n}}^-$$

where $\mathbf{v}$ is a generic vector and w is a generic scalar. Note that the jump of a vector is a scalar while the jump of a scalar is a vector. Furthermore, note that the jump will be zero for a continuous function. Now it is possible to discretize an equation using DG, as follows in the next section.

3.2 DG formulation for advection problems

Consider the advection of a scalar quantity u with flux $\mathbf{F}(u)$ and source term $S(u)$ satisfying

$$\begin{aligned} \frac{\partial u}{\partial t} + \nabla \cdot \mathbf{F}(u) &= S(u), \quad \text{in } (0,T) \times \Omega && (3.6) \\ u &= g_D, \quad \text{on } \partial\Omega_D && (3.7) \\ \mathbf{F}(u) \cdot \hat{\mathbf{n}} &= g_N, \quad \text{on } \partial\Omega_N && (3.8) \\ u &= u_0, \quad \text{in } (0,0) \times \Omega && (3.9) \end{aligned}$$

over domain Ω from time 0 to time T, where g_D is the value of the Dirichlet boundary on $\partial\Omega_D$ and g_N is the value of the Neumann boundary on $\partial\Omega_N$. Let $\mathcal{P}^p(\Gamma)$ denote the set of polynomials of degree at most p on a domain Γ. Discretizing the domain with triangulation $\mathcal{T}_h$ of non-overlapping elements $\mathcal{T}_h = \bigcup_{i=1}^{N_t} K_i$ where N_t is the number of triangles, we seek an approximation u_h of u with $u_h \in W_h^p$ where

$$W_h^p = \left\{ w \in L^2(\Omega) : w \mid_K \in \mathcal{P}^p(K), \forall K \in \mathcal{T}_h \right\} \tag{3.10}$$

such that:

$$\int_K \left\{ \frac{\partial u_h}{\partial t} w + \left[\nabla \cdot \mathbf{F}(u_h) \right] w \right\} dK = \int_K S(u_h) w dK, \quad \forall K \in \mathcal{T}_h \tag{3.11}$$

For readers unfamiliar with the notation, equation 3.10 reads: take w to lie in the L^2 space that exists on Ω such that w restricted to an element K lies in the polynomial space $\mathcal{P}^p$ that exists on K. Equation 3.11 is not a complete DG formulation, since currently the solutions on individual elements are not coupled. Following an approach similar to the FV method, we integrate the advection term by parts

$$\begin{aligned} \int_K \frac{\partial u_h}{\partial t} w dK + \int_K \nabla \cdot \left[\mathbf{F}(u_h) w \right] dK - \int_K \mathbf{F}(u_h) \cdot \nabla w dK &= \int_K S(u_h) w dK \\ \int_K \frac{\partial u_h}{\partial t} w dK + \int_{\partial K} \hat{\mathbf{F}}(u_h) \cdot \hat{\mathbf{n}} w d\partial K - \int_K \mathbf{F}(u_h) \cdot \nabla w dK &= \int_K S(u_h) w dK \end{aligned} \tag{3.12}$$

where the second step follows from the Divergence theorem. Note that the notation $\hat{\mathbf{F}}(u_h)$ indicates that the solution on the edge bordering two elements is a function of both the bordering elements, thereby achieving a coupling between elements. In order to satisfy conservation, the value of $\hat{\mathbf{F}}(u_h)$ is taken as constant on an edge, that is, the two bordering elements will use the *same* value of $\hat{\mathbf{F}}(u_h)$. Equation 3.12, then gives the weak formulation of 3.6, and the scheme will be complete as soon as the functional form of $\hat{\mathbf{F}}(u_h)$ is specified. Alternatively, a strong formulation for the problem can be found by taking an additional integration by parts in equation 3.12 as follows

$$\int_K \frac{\partial u_h}{\partial t} w dK + \int_{\partial K} \left[\hat{\mathbf{F}}(u_h) - \mathbf{F}(u_h)\right] \cdot \hat{\mathbf{n}} w d\partial K + \int_K \left[\nabla \cdot \mathbf{F}(u_h)\right] w dK = \int_K S(u_h) w dK \tag{3.13}$$

where the second application of the Divergence theorem uses $\mathbf{F}(u_h)$ instead of $\hat{\mathbf{F}}(u_h)$ to obtain a unique formulation (otherwise we recover 3.11). While 3.12 and 3.13 are mathematically equivalent, their numerical implementations are different, and there are some advantages in terms of implementation and efficiency using one form over the other for some problems. Also, after a re-arrangement of 3.13

$$\int_K \left\{ \frac{\partial u_h}{\partial t} + \left[\nabla \cdot \mathbf{F}(u_h)\right] - S(u_h) \right\} w dK = \int_{\partial K} \left[\mathbf{F}(u_h) - \hat{\mathbf{F}}(u_h)\right] \cdot \hat{\mathbf{n}} w d\partial K$$

it is highlighted that the residual on the borders of an element serves to couple elements within a triangulation.

The $\hat{\mathbf{F}}(u_h)$ term is not present in CG FEM discretizations, but for DG schemes, the proper specification of $\hat{\mathbf{F}}(u_h)$ can stabilize the numerical scheme. The problem with CG FEM discretizations when it comes to advective problems is that CG schemes inherently use a "central" difference type discretization for the flux. While this is more accurate than an upwind scheme, it is well known that central schemes tend to be unstable (Chapra and Canale, 2006) for advective problems. To stabilize the advection scheme, then, a number of strategies can be employed, but all involve

adding some numerical dissipation to the scheme. With CG, adding the dissipation can be a complicated process, but with DG, this dissipation can very naturally be added through the $\hat{\mathbf{F}}(u_h)$ flux terms. In order to choose an appropriate functional form for $\hat{\mathbf{F}}(u_h)$ that adds the minimum amount of dissipation to the scheme, we make use of the results from the well-studied Riemann problem.

3.2.1 Riemann solvers for DG

This section makes extensive use of chapters 2 and 6 of Hesthaven and Warburton (2008) and the excellent text by LeVeque (2002). This section only serves as a brief review, and the reader is referred to LeVeque (2002) for further study.

The Riemann problem is named after Bernhard Riemann, and it involves the solution of a conservation law together with piecewise constant initial conditions containing a single discontinuity. The Riemann problem is useful for understanding hyperbolic systems of equations, because all the properties (such as shocks and rarefaction waves for the Euler equations) appear as characteristics, or "Riemann invariants" in the solution of the Riemann problem. When solving conservations laws in the DG framework, the discontinuity arises at the interface of two elements, where a jump in the value of the properties occur, and theory from the Riemann problem is used to construct the fluxes properly.

A general, non-linear hyperbolic system of equations in two-dimensions can be written as

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}_x(\mathbf{u})}{\partial x} + \frac{\partial \mathbf{f}_y(\mathbf{u})}{\partial y} = 0 \tag{3.14}$$

and in the DG context we are interested in finding an approximation for $\hat{\mathbf{F}}(\mathbf{u}) \cdot \hat{\mathbf{n}} = \mathbf{f}_x \hat{n}_x + \mathbf{f}_y \hat{n}_y$. Because we are only interested in a one-dimensional flux normal to the boundary, we can make use of the theory for linearized hyperbolic one-dimensional

systems. The above system is rewritten as

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}_x(\mathbf{u})}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial x} + \frac{\partial \mathbf{f}_y(\mathbf{u})}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial y} &= 0 \\ \frac{\partial \mathbf{u}}{\partial t} + \mathbf{A}_x\frac{\partial \mathbf{u}}{\partial x} + \mathbf{A}_y\frac{\partial \mathbf{u}}{\partial y} &= 0 \end{aligned}$$

by using the chain-rule and letting $\mathbf{A}_x$ and $\mathbf{A}_y$ be the $d \times d$ Jacobian matrices, where d is the dimension of the problem. Now we use

$$\hat{\mathbf{A}} = \mathbf{A}_x\hat{n}_x + \mathbf{A}_y\hat{n}_y$$

and we can consider the one-dimensional system

$$\frac{\partial \mathbf{u}}{\partial t} + \hat{\mathbf{A}}\frac{\partial \mathbf{u}}{\partial \hat{\mathbf{n}}} = 0 \tag{3.15}$$

where $\hat{\mathbf{A}}$ is a function of $\mathbf{u}$. Now, hyperbolic systems of equations are characterized by the fact that $\hat{\mathbf{A}}$ is diagonalizable, that is

$$\hat{\mathbf{A}} = \mathbf{S}\Lambda\mathbf{S}^{-1} \tag{3.16}$$

$$|\hat{\mathbf{A}}| = \mathbf{S}|\Lambda|\mathbf{S}^{-1} \tag{3.17}$$

where we have also defined $|\hat{\mathbf{A}}|$. Here Λ is a diagonal matrix with the eigenvalues on the diagonals, and the columns of $\mathbf{S}$ contain the eigenvectors of A. Multiplying equation 3.15 by $\mathbf{S}^{-1}$ and setting $\mathbf{S}^{-1}\mathbf{u} = \mathcal{I}$ we have

$$\frac{\partial \mathcal{I}}{\partial t} + \Lambda\frac{\partial \mathcal{I}}{\partial \hat{\mathbf{n}}} = 0 \tag{3.18}$$

where the entries of $\mathcal{I}$ are termed the "Riemann invariants." Through this procedure, one obtains a decoupled system of equations, where coupling remains only through the eigenvalues of the system. Each scalar invariant $\mathcal{I}_j$ is advected at the speed λ_j, where the speed is in the normal direction if $\lambda_j > 0$ and the speed is opposite the normal direction when $\lambda_j < 0$. According to the theory of characteristics, the following is the

solution for an initially discontinuous state if using $\hat{\mathbf{n}} = \hat{\mathbf{n}}^+$ and referring to Figure 3-4:

$$\mathcal{I}_j = \mathcal{I}_j^+ \quad \text{if} \lambda_j > 0 \tag{3.19}$$

$$\mathcal{I}_j = \mathcal{I}_j^- \quad \text{if} \lambda_j < 0 \tag{3.20}$$

With some manipulation (Hesthaven and Warburton, 2008) we can recover the form for the flux

$$\hat{\mathbf{F}}(\mathbf{u}) \cdot \hat{\mathbf{n}}^+ = \hat{\mathbf{A}}\{\{\mathbf{u}\}\} + \frac{1}{2}|\hat{\mathbf{A}}|[\![\mathbf{u}]\!] \tag{3.21}$$

where $\hat{\mathbf{A}}$ is a function of both $\mathbf{u}^+$ and $\mathbf{u}^-$ for general non-linear fluxes. For linear flux functions, the formulation is complete since $\mathbf{A}$ would not be a function of $\mathbf{u}^{\pm}$. For non-linear fluxes, what remains is to choose the form for $\hat{\mathbf{A}}$ and $|\hat{\mathbf{A}}|$, and this distinguishes the various approximate Riemann solvers from each other. Note that $\hat{\mathbf{A}}\mathbf{u} \approx \hat{\mathbf{F}}(\mathbf{u}) \cdot \hat{\mathbf{n}}$ is a linearization of the flux. A natural choice yielding a consistent flux is to let

$$\hat{\mathbf{F}}(\mathbf{u}) \cdot \hat{\mathbf{n}}^+ = \{\{\mathbf{F}(\mathbf{u}) \cdot \hat{\mathbf{n}}^+\}\} + \frac{1}{2}|\hat{\mathbf{A}}|[\![\mathbf{u}]\!] \tag{3.22}$$

while appropriately choosing a form for $|\hat{\mathbf{A}}|$. Two possible choices are

$$|\hat{\mathbf{A}}| = |\mathbf{A}_x(\{\{\mathbf{u}\}\})\hat{n}_x + \mathbf{A}_y(\{\{\mathbf{u}\}\})\hat{n}_y| \tag{3.23}$$

$$|\hat{\mathbf{A}}| = |\{\{\mathbf{A}_x(\mathbf{u})\}\}\hat{n}_x + \{\{\mathbf{A}_y(\mathbf{u})\}\}\hat{n}_y| \tag{3.24}$$

where care needs to be taken to ensure that $|\hat{\mathbf{A}}|$ has purely real eigenvalues for 3.24.

An often used approximate Riemann solver (due to its ease of implementation) is the local Lax-Friedrichs solver, which assumes that there is one dominating wave in the system, and enough numerical dissipation is added to stabilize the scheme for

this wave. The Lax-Friedrichs solver is as follows

$$\hat{\mathbf{F}}(\mathbf{u}) \cdot \hat{\mathbf{n}}^+ = \{\{\mathbf{F}(\mathbf{u}) \cdot \hat{\mathbf{n}}^+\}\} + \frac{1}{2}|\lambda_{max}|[\![\mathbf{u}]\!] \tag{3.25}$$

where λ_{max} is the eigenvalue with the largest magnitude. The local Lax-Friedrichs solver chooses the value of λ_{max} locally on an edge.

3.2.2 Quadrature-free versus Quadrature based algorithms

Consider the numerical implementation of:

$$\int_K \mathbf{F}(u) \cdot \nabla w dK \approx \int_K \mathbf{F}\left(\sum_i u_{h,i}\theta_i(\mathbf{x})\right) \cdot \nabla\theta_j(\mathbf{x}) dK \tag{3.26}$$

or

$$\int_K \mathbf{F}(u) \cdot \nabla w dK \approx \int_K \sum_i \mathbf{F}(u_{h,i})\theta_i(\mathbf{x}) \cdot \nabla\theta_j(\mathbf{x}) dK \tag{3.27}$$

Clearly, there are two choices for discretizing the equation. For the first case (3.26) a quadrature scheme is introduced to perform the volume integral

$$\int_K \mathbf{F}(u) \cdot \nabla w dK \approx \sum_k^{N_{gp}} \mathbf{F}\left(\sum_i u_{h,i}\theta_i(\mathbf{x}_k)\right) \cdot \nabla\theta_j(\mathbf{x}_k)\mathcal{W}_k J(\mathbf{x}_k) \tag{3.28}$$

where $\mathcal{W}$ are the gauss weights and $J(\mathbf{x})$ is a Jacobian (for the coordinate transformation between the master and current element) evaluated at gauss point $\mathbf{x}$. For a non-linear flux $\mathbf{F}$, this scheme cannot be further simplified, and this integration needs to be performed for every element. The total cost involves a series of matrix-vector multiplies to interpolate the nodal/modal values onto the gauss points, followed by another matrix-vector multiply to perform the integration. Also, normally the number of gauss weights are taken to be greater than the number of nodes or modes, resulting in larger matrices and a larger computational expense.

For the second choice of discretization, 3.27, we can write

$$\begin{aligned}\int_K \mathbf{F}(u)\cdot\nabla w dK &\approx F(u_{h,i})\int_K \sum_i \theta_i(\mathbf{x})\cdot\nabla\theta_j(\mathbf{x})dK \\ &= \mathbf{C}^k\mathbf{F}(u_{h,i}) \end{aligned} \tag{3.29}$$

where $\mathbf{C}^k_{ji} = \int_K \theta_i(\mathbf{x})\cdot\nabla\theta_j(\mathbf{x})dK$ and can be precomputed. What remains is to calculate the fluxes at the nodal points, or for each mode. Straight-sided triangles have a constant Jacobian, in which case $\mathbf{C}$ can be precomputed for a reference element and scaled with J to form $\mathbf{C}^k$. In that case, no quadrature scheme is required, which means only an evaluation of the fluxes and a single matrix-vector multiplication is required. This is significant saving over the quadrature scheme. As an additional benefit, the $\mathbf{M}^{-1}\mathbf{C}$ matrix can be precomputed, which results in additional savings. However, the quadrature-free scheme commits a variational crime, which can result in problems. For instance, the non-linear fluxes may suffer from aliasing errors causing instabilities in the solution, but these may be remedied by some minor filtering of the higher modes (Hesthaven and Warburton, 2008) at slightly reduced accuracy and rate of convergence compared to the quadrature-based scheme. For a discussion on the magnitude of the aliasing errors, which depend on the smoothness of u_h and $\mathbf{F}(u_h)$, the reader is referred to Chapter 5 of Hesthaven and Warburton (2008).

3.3 DG with second order derivatives

The extension of DG to higher order derivatives is discussed in this section. Only the extension to second order equations will be discussed in detail, and for a generalization to higher orders, the reader is referred to Yan and Shu (2002). We could consider the problem

$$\frac{\partial u}{\partial t} + \nabla\cdot\mathbf{F}_{inv}(u) + \nabla\cdot\mathbf{F}_{vis}(u,\ \nabla u) = S(u) \tag{3.30}$$

where $F_{inv}(u)$ is the functional form for the inviscid fluxes, and $\mathbf{F}_{vis}(u,\ \nabla u)$ is the functional form for the viscous fluxes. However, since the advection and source term have already been discretized in Section 3.2, we shall consider the simpler equation

$$\frac{\partial u}{\partial t} + \nabla \cdot [-\kappa \nabla u] \ = \ 0 \tag{3.31}$$

where we have taken $\mathbf{F}_{vis}(u,\ \nabla u) = -\kappa \nabla u$ with κ a constant or some function of $\mathbf{x}$.

A simple choice for the numerical flux $\hat{\mathbf{F}}_{vis}$ is obtained by calculating the derivative of the solution within each element and then using a central flux, that is $\hat{\mathbf{F}}_{vis} = \{\{\frac{\kappa \partial \mathbf{u}}{\partial \hat{\mathbf{n}}}\}\}$. However, while the discrete matrix is relatively well conditioned, this scheme was proven to yield unstable results in some cases, for example see chapter 7 of Hesthaven and Warburton (2008). Little progress was made until Bassi and Rebay (1997) suggested rewriting the equation as a system of first-order equations, in which case 3.31 becomes:

$$\begin{aligned}
\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{q} &= 0, \quad \text{in } (0,T) \times \Omega && (3.32)\\
\mathbf{q} + \kappa \nabla u &= 0, \quad \text{in } \Omega && (3.33)\\
u &= g_D, \quad \text{on } \partial\Omega_D && (3.34)\\
\mathbf{q} \cdot \hat{\mathbf{n}} &= g_N, \quad \text{on } \partial\Omega_N && (3.35)\\
u &= u_0, \quad \text{in } (0,0) \times \Omega && (3.36)
\end{aligned}$$

The discretization of 3.32 proceeds the same as the discretization of 3.6 using the same space W_h as defined in 3.10, such that we have

$$\begin{aligned}
\int_K \frac{\partial u_h}{\partial t} w dK + \int_{\partial K} \hat{\mathbf{q}_h} \cdot \hat{\mathbf{n}} w d\partial K - \int_K \mathbf{q}_h \cdot \nabla w dK &= 0 && (3.37)\\
\int_K \frac{\partial u_h}{\partial t} w dK + \int_{\partial K} [\hat{\mathbf{q}_h} - \mathbf{q}_h] \cdot \hat{\mathbf{n}} w d\partial K + \int_K \nabla \cdot \mathbf{q}_h w dK &= 0 && (3.38)
\end{aligned}$$

in the weak (3.37) and strong (3.38) forms. The discretization of 3.33 proceeds simi-

larly, but first we need to define a new space V_h^p

$$V_h^p = \left\{ \mathbf{v} \in (L^2(\Omega))^d : \mathbf{v} \mid_K \in (P^p(K))^d, \forall K \in \mathcal{T}_h \right\} \tag{3.39}$$

where d is the dimension of the problem. V_h^p is a vector space, which is different from the scalar space W_h^p defined in 3.10. Then proceeding to discretize 3.33 we have

$$\begin{aligned}
\int_K \mathbf{q}_h \cdot \mathbf{v} dK + \int_K \kappa \nabla u_h \cdot \mathbf{v} dK &= 0 \\
\int_K \kappa^{-1} \mathbf{q}_h \cdot \mathbf{v} dK + \int_{\partial K} \hat{u}_h \mathbf{v} \cdot \mathbf{n} d\partial K - \int_K u_h \nabla \cdot \mathbf{v} dK &= 0 \qquad (3.40) \\
\int_K \kappa^{-1} \mathbf{q}_h \cdot \mathbf{v} dK + \int_{\partial K} [\hat{u}_h - u_h] \mathbf{v} \cdot \mathbf{n} d\partial K + \int_K \nabla u_h \cdot \mathbf{v} dK &= 0 \qquad (3.41)
\end{aligned}$$

in the weak (3.40) and strong (3.41) forms. What is left is to specify the form for the numerical fluxes. The most general form of the flux (which follows from a stability analysis) as written using the notation suggested by Castillo et al. (2000), is as follows:

$$\begin{aligned}
\hat{\mathbf{q}}_\mathbf{h} &= \{\{\mathbf{q}_h\}\} - C_{11} [\![u_h \hat{\mathbf{n}}]\!] + \mathbf{C}_{12} [\![\mathbf{q}_h \cdot \hat{\mathbf{n}}]\!] \qquad (3.42) \\
\hat{u}_h &= \{\{u_h\}\} - \mathbf{C}_{12} \cdot [\![u_h \hat{\mathbf{n}}]\!] - C_{22} [\![\mathbf{q}_h \cdot \hat{\mathbf{n}}]\!] \qquad (3.43)
\end{aligned}$$

The values chosen for the parameters C_{11}, $\mathbf{C}_{12}$, and C_{22} results in a number of different schemes for elliptic problems using a DG discretization. Castillo et al. (2000) also showed that $C_{11} > 0$ and $C_{22} \geq 0$ is required for the DG method to give a unique approximate solution. Arnold et al. (2002) analyzed some of the available schemes under a unified framework, and more recent work by Cockburn et al. (2009) analyzes the existing schemes under the new Hybrid Discontinuous Galerkin (HDG) framework, which unifies not only DG methods for second order elliptic problems, but also CG and mixed-methods. The following sections briefly describe some of the popular methods for discretizing elliptic problems using DG.

Unless explicitly stated, the schemes described treat the boundary conditions in a weak sense. That is, to satisfy the boundary conditions for the set of equations

3.32-3.36 the following is used:

$$\hat{u}_h = u_D, \quad \text{on} \quad \partial\Omega_D \tag{3.44}$$

$$\hat{\mathbf{q}}_\mathbf{h} = \mathbf{q}_h - C_{11}(u_h - u_D)\hat{\mathbf{n}}, \quad \text{on} \quad \partial\Omega_D \tag{3.45}$$

$$\hat{u}_h = u_h, \quad \text{on} \quad \partial\Omega_N \tag{3.46}$$

$$\hat{\mathbf{q}}_\mathbf{h} = g_N\hat{\mathbf{n}}, \quad \text{on} \quad \partial\Omega_N \tag{3.47}$$

Also, the majority of schemes take $C_{22} = 0$. This serves to decouple the solution of u and $\mathbf{q}$, which means $\mathbf{q}$ can be solved independently of u, thereby allowing the scheme to be less computationally expensive. The alternative is to solve for u and $\mathbf{q}$ simultaneously which is a strategy employed by some variants of the Local Discontinuous Galerkin (LDG) method, and the HDG methods.

3.3.1 Internal Penalty (IP) method

The IP method for discontinuous elements were originally developed by Arnold (1982), and uses the following

$$\mathbf{C}_{12} = 0, \quad C_{11} = \tau, \quad C_{22} = 0$$

$$\hat{\mathbf{q}}_\mathbf{h} = \{\{\nabla u_h\}\} - \tau[\![u_h\hat{\mathbf{n}}]\!] \tag{3.48}$$

where τ is chosen appropriately according to the application. With this choice of parameters, $\hat{\mathbf{q}}_h$ is penalized by τ times the jump in u_h, and $\hat{u}_h$ uses the average value of u_h on the edge. The IP method combines sparsity with a low condition number (comparable to the condition number when using central fluxes (Hesthaven and Warburton, 2008)) for the discrete operator, which makes IP a popular method.

3.3.2 The Local Discontinuous Galerkin (LDG) method

The LDG method was introduced by Cockburn and Shu (1998a), and uses the following:

$$\mathbf{C}_{12} = \frac{\mathbf{n}^{\pm}}{2}, \quad C_{11} = \tau, \quad C_{22} = 0 \tag{3.49}$$

With this choice of parameters, $\hat{\mathbf{q}}_h$ is penalized by τ times the jump in u_h and $\frac{\mathbf{n}^{\pm}}{2}$ times the jump in $\mathbf{q}_h$, and $\hat{u}_h$ is penalized by $\frac{\mathbf{n}^{\pm}}{2}$ the jump in u_h. Note that $\mathbf{C}_{12}$ can be arbitrarily chosen to be either associated with $\hat{\mathbf{n}}^+$ or $\hat{\mathbf{n}}^-$, the only restriction being that $\mathbf{C}_{12} \neq \hat{\mathbf{n}}^+ \forall \partial K^+ \in K^+$. That is, at least one of the edges in the element has to associate $\mathbf{C}_{12}$ with a different normal than the other edges. If this criteria is not satisfied, the scheme can still be stable for non-zero τ, but if it is satisfied the scheme is stable for $\tau = 0$ and gives the minimum dissipation scheme (Hesthaven and Warburton, 2008). Normally τ is chosen to be zero in the interior, and non-zero on the boundary of the domain. A larger value of τ on the boundary enforces the boundary conditions more strictly, where in the limit of $\tau = \infty$ the boundary conditions are enforced in a strong sense.

Using the LDG fluxes, we can rewrite $\hat{u}_h = \{\{u_h\}\} - \mathbf{C}_{12}[\![u\hat{\mathbf{n}}]\!]$ and $\hat{\mathbf{q}}_\mathbf{h} = \{\{\mathbf{q}_h\}\} - C_{11}[\![u_h\hat{\mathbf{n}}]\!] + C_{12}[\![\mathbf{q}\cdot\hat{\mathbf{n}}]\!]$ in a simpler form as follows:

$$\begin{aligned}
\hat{u}_h &= \frac{u_h^+ + u_h^-}{2} - \frac{\mathbf{n}^{\pm}}{2}\cdot[u_h^+\mathbf{n}^+ - u_h^-\mathbf{n}^+] \\
&= \frac{u_h^+ + u_h^-}{2} \mp \frac{u_h^+ - u_h^-}{2} \\
&= \frac{(-\tilde{C}_{12}+1)}{2}u_h^+ - \frac{(-\tilde{C}_{12}-1)}{2}u_h^- \qquad (3.50) \\
\hat{\mathbf{q}}_\mathbf{h} &= \frac{\mathbf{q}_h^+ + \mathbf{q}_h^-}{2} - C_{11}[\![u_h\hat{\mathbf{n}}]\!] + \frac{\mathbf{n}^{\pm}}{2}[\mathbf{q}_h^+\cdot\mathbf{n}^+ - \mathbf{q}_h^-\mathbf{n}^+] \\
&= \frac{(\tilde{C}_{12}+1)}{2}\mathbf{q}_h^+ - \frac{(\tilde{C}_{12}-1)}{2}\mathbf{q}_h^- - C_{11}[\![u_h\hat{\mathbf{n}}]\!] \qquad (3.51) \\
\tilde{C}_{12} &= \pm 1 \qquad (3.52)
\end{aligned}$$

This form highlights the "flip-flop" nature of the scheme, that is, if $\hat{u}_h$ is chosen from

K^+, $\hat{\mathbf{q}}$ is chosen from K^-. The criteria for enabling a stable minimum dissipation scheme for $\tilde{C}_{12}$ is then

$$\left| \sum_{\partial K} \tilde{C}_{12} \right| < N_e \quad \forall K \in \mathcal{T}_h \tag{3.53}$$

where N_e is the number of edges in the element. With a clever implementation, $\mathbf{C}_{12}$ can be taken as $\mathbf{C}_{12} = 1$ (or $= -1$) on all the *edges* (as defined globally) while still enforcing the criteria to obtain the minimum dissipation scheme.

The minimum dissipation property makes LDG an attractive scheme for solving second order elliptic equations using DG, however unlike the IP scheme, for example, it has a non-compact stencil in higher dimensions, which means it takes information from elements that are not direct neighbours. The problem can be overcome by slightly modifying the fluxes, which is achieved by using the Compact Discontinuous Galerkin (CDG) method described next. Finally, the condition number of the discrete global operator is approximately twice as large as the condition number for the IP method (Hesthaven and Warburton, 2008).

3.3.3 The Compact Discontinuous Galerkin (CDG) method

The CDG method is a modification of the LDG method developed by Peraire and Persson (2007). A compact stencil is achieved in CDG by carefully studying how the non-compactness of the $\hat{\mathbf{q}}_h$ fluxes arise in LDG. Referring to Figure 3-5 the non-compactness of the LDG method can be explained.

Consider element 1. On the edge a, $\hat{u}_h$ is taken as u_h^1 for calculating $\mathbf{q}_h^1$ and $\mathbf{q}_h^2$. However, on edge b, $\hat{u}_h$ is taken as u_h^3, which means the calculation of $\mathbf{q}_h^2$ also contains information from element 3. Due to the flip-flop nature of LDG, on edge a, $\hat{\mathbf{q}}_h$ is then taken as $\mathbf{q}_h^2$ which contains information from element 3, thereby resulting in a non-compact scheme. The CDG method recognizes this situation, and to remedy the problem it saves a version of $\mathbf{q}_h^2$ that does not include the information from element 3.

The fluxes are, then, the same as the LDG fluxes, except that $\mathbf{q}_h$ is replaced by $\mathbf{q}_{e,h}$

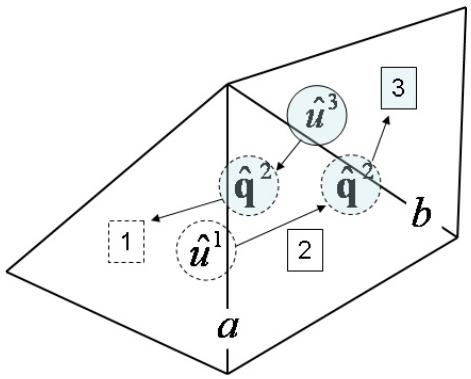

Figure 3-5: Non-compactness of LDG in multiple dimensions.

which contains only information from neighboring elements, resulting in a compact stencil:

$$\begin{aligned} \mathbf{C}_{12} &= \frac{\mathbf{n}^{\pm}}{2}, \quad C_{11} = \tau, \quad C_{22} = 0 \\ \hat{\mathbf{q}_h} &= \{\{\mathbf{q}_{e,h}\}\} - C_{11}[\![u_h \hat{\mathbf{n}}]\!] + \mathbf{C}_{12}[\![\mathbf{q}_{e,h} \cdot \hat{\mathbf{n}}]\!] \end{aligned} \tag{3.54}$$

The flux $\hat{\mathbf{q}_h}$ can be written in terms of $\tilde{C}_{12}$ as

$$\hat{\mathbf{q}_h} = \frac{(\tilde{C}_{12}+1)}{2}\mathbf{q}^+_{e,h} - \frac{(\tilde{C}_{12}-1)}{2}\mathbf{q}^-_{e,h} - C_{11}[\![u_h \hat{\mathbf{n}}]\!] \tag{3.55}$$

CDG retains all the theoretical properties of LDG, and numerical experiments have shown that the stability for CDG may even be enhanced compared to the stability of LDG. The main advantage of CDG is the more compact stencil for efficient numerical treatment of implicit time integration schemes, at the cost of slightly more expensive flux evaluations for matrix-free iterative methods.

3.3.4 Hybrid Discontinuous Galerkin (HDG) method

The HDG method is the newest method discussed, with Cockburn et al. (2009), Nguyen et al. (2009) being the main references in the literature. The derivation of HDG methods are significantly different from the previous methods, but can be recast

in the more standard form as follows:

$$C_{11} = \frac{\tau^+\tau^-}{\tau^+ + \tau^-}, \quad \mathbf{C}_{12} = \frac{1}{2}\left(\frac{[\![\tau\hat{\mathbf{n}}]\!]}{\tau^+ + \tau^-}\right), \quad C_{22} = \frac{1}{\tau^+ + \tau^-} \tag{3.56}$$

where the parameter τ is non-uniquely defined on each edge. With this choice of parameters, $\hat{\mathbf{q}}_h$ is penalized by both the jump in u_h and $\mathbf{q}_h$, and $\hat{u}_h$ is also penalized by both the jump in u_h and $\mathbf{q}_h$.

The derivation of the HDG scheme is as follows. First we define $e = \partial K^+ \bigcap \partial K^-$ in the interior and $e = \partial K \bigcap \partial\Omega$ on the boundary. Then we define a new space:

$$M_h^p = \{\mu \in L^2(\varepsilon_h) : \mu|_e \in \mathcal{P}^p(e), \forall e \in \varepsilon_h\} \tag{3.57}$$

That is, the space M_h^p is continuous on each edge of ε_h (as opposed to V_h^p and W_h^p which are discontinuous). HDG is derived by noting that each element can be solved independently of the other triangles if the value at the boundary of the element $\hat{u}$ is known. The boundary data $\hat{u}$ can be expressed in terms of a new variable that we define as $\lambda_h \in M_h^p(0)$, where the notation $M_h^p(0)$ refers to the space M_h^p that is zero-valued on the boundaries of the domain:

$$\hat{u} = \begin{cases} Pg_D, & \text{on } \varepsilon_h^\partial \\ \lambda_h, & \text{on } \varepsilon_h^\circ \end{cases} \tag{3.58}$$

Here P is an operator that projects the boundary data, g_D, onto $\hat{u}$. Note that the boundary conditions for HDG are enforced strongly for $\hat{u}$, but this translates to a weak enforcement inside the elements. The form for the flux $\hat{\mathbf{q}}_h$ is then chosen as in Nguyen et al. (2009)

$$\hat{\mathbf{q}}_h = \mathbf{q}_h + \tau(u_h - \hat{u}_h) \tag{3.59}$$

where τ can be taken as $\tau = \mathcal{O}(1)$ for optimal convergence of elliptic problems. This flux definition is not unique, but this choice gives convergent, stable results for properly chosen values of τ. This allows the local problems to be written completely

in terms of the boundary data $\hat{u}_h$. The boundary data $\hat{u}_h$ then needs to be solved as a coupled set of equations that enforce conservativity of the fluxes between elements. The global equation solved is then

$$\int_{\varepsilon_h} [\![\hat{\mathbf{q}}]\!] d\varepsilon_h = \int_{\varepsilon_h^{\partial}} g_N \mu d\Gamma_N \tag{3.60}$$

This solution method involves three steps:

1. The inversion of a local operator on each element to form the right-hand-side vector (and global matrix)

2. The global solution of the boundary data

3. The local reconstruction of the solution on the element

The local operations are cheap because inversions are done on small dense matrices, while the global solve contains considerably less unknowns than the original system that would be obtained from an IP or CDG method. This procedure dramatically increases the efficiency of solving elliptic problems with DG where implicit time integration is required. Implicit time integration may be necessary to overcome stringent numerical stability criteria which limit the timestep size for explicit schemes.

As an additional benefit with this method, both u_h and $\hat{\mathbf{q}}_h$ converge at the optimal rate of $\mathcal{O}(p+1)$ when $\tau = \mathcal{O}(1)$ which allows a post-processed solution u_h^* which converges at $\mathcal{O}(p+2)$. This property is lost for large values of τ. The post processing can be achieved by solving the following diffusion equation locally on each element:

$$\int_K \kappa \nabla u_h^* \cdot \nabla w^* dK = -\int_K \mathbf{q} \cdot \nabla w^* dK, \quad \forall w^* \in \mathcal{W}_h^{p+1} \tag{3.61}$$

$$\int_K u_h^* dK = \int_K u_h dK \tag{3.62}$$

For additional details about HDG, the reader is referred to Nguyen et al. (2009).

3.4 Implementation issues

Implementation issues related to the non-linear fluxes have already been discussed in section 3.2.2. The remaining issues include the data-structures (see Persson and Peraire (2006)) which will not be discussed, the solution method for inverting large matrices which is the topic of the Chapter 5, and the type of basis to use, which is the topic of this section.

An example of a simple modal basis set in one-dimension of order p is the monomial basis x^{p-1}. Here the unknown coefficients are related to each mode, and are not directly related to the solution.

Another popular basis set is the nodal basis, which is related to a set of $p+1$ points $\mathbf{x}$ for a p^{th} order basis set. An example of a nodal basis in one-dimension is the Lagrange polynomial $\ell_i(r) = \prod_{j=1, j\neq i}^{p+1} \frac{r-x_j}{x_i-x_j}$ that has the property $\ell_i(x_i) = \delta_{ij}$, where r is an arbitrary point. That is, the basis related to point x_i is equal to one at x_i and zero at all the other points $\mathbf{x} \neq x_i$.

There are some advantages and disadvantages to each approach (Karniadakis and Sherwin, 2005). The modal approach handles p-adaptivity (adapting the order of the basis function) more naturally, but requires a function evaluation of all the bases to find the value of u_h at any point in the element. The nodal approach has the advantage that the expansion coefficients are equal to the approximate value of the function at the specified nodal points $\mathbf{x}$. Thus, to determine the value of u_h at the nodal points, no function evaluation is needed and simply the coefficient needs to be read from memory.

For the purposes of this work, a nodal basis will be used. The details of creating this nodal basis is discussed in Hesthaven and Warburton (2008), and a similar implementation is used for this work. The procedure is somewhat complicated because a close-form analytical solution for a nodal basis on a triangle does not exist. That is, given a set of nodes and the order of a basis, an analytical expression describing the shape of all the basis functions has not been found. In order to construct the discrete operators for a nodal basis, an appropriate modal basis is used.

The reason why this works is because the modal basis and nodal basis can be related to each other due to the uniqueness of the polynomial representation, that is $\sum_{i=1}^{N} u_{h,i}(t)\psi_i(\mathbf{x}) = \sum_{i=1}^{N} u_{h,i}(t, \mathbf{x}_i)\phi_i(\mathbf{x})$.

The overall procedure for using a nodal basis to construct the FEM operators as follows:

1. Solve for modal coefficient which relate the modal and nodal bases

2. Evaluate the modal basis at specified points and using the modal coefficients from step 1, find the values of the nodal basis at the specified points

3. Evaluate the derivatives of the modal basis at specified points and using the modal coefficients from step 1, find the values of the derivatives of the nodal basis at the specified points

4. Construct the finite-element operators, such as the mass matrix, using a combination of steps 1, 2, and 3. Save the constructed matrices for later use.

First, the coefficients which related the modal and nodal basis functions are found. In order to do this, the modal Koornwinder basis (Koornwinder, 1991), which is an orthogonal polynomial basis constructed using Jacobi polynomials, is used to find the coefficients $u_{h,i}(t, \mathbf{x}_i)$. In what follows we use u^M and u^N to differentiate between the coefficients for the modal basis and nodal basis respectively. Then

$$\mathcal{V} u_i^M = u_i^N \tag{3.63}$$

where $\mathcal{V}$ is the Vandermonde matrix (see Trefethen and Bau (1997)) with $\mathcal{V}_{ij} = \psi_j(x_i)$, that is, the j^{th} modal function evaluated at the i^{th} nodal point. We know that we want the value of the i^{th} nodal basis to be 1 at the i^{th} nodal point (x_i), and zero for all the other nodal points $\mathbf{x} \neq \mathbf{x}_i$. We can then solve for all the modes that will give this polynomial basis

$$u_i^M = \mathcal{V}^{-1} u_i^N \tag{3.64}$$

where u_i^N is a zero vector except for a 1 at the i^{th} entry.

Second, with this modal polynomial basis in place and the coefficients relating the two bases, the value of the nodal basis can be evaluated at specific points. All that is required is the value of the modal basis at the desired points and the solved values of the modal coefficients u^M, from the first step. The values of the nodal basis can be found by multiplying through with another Vandermonde matrix

$$\mathcal{V}_{gpts} u_i^M = u_{i,gpts}^N \tag{3.65}$$

$$\mathcal{V}_{gpts} \mathcal{V}^{-1} \mathbf{I} = \mathcal{V}_{N,gpts} \tag{3.66}$$

where $\mathcal{V}_{gpts}$ is the Vandermonde matrix for that modal basis that evaluates the function at, for example, the gauss points $\mathbf{x}_{i,gpts}$, and $\mathcal{V}_{N,gpts} = \phi_j(x_{i,gpts})$ is the Vandermonde matrix for the nodal basis.

Third, we can similarly get the value of the derivatives of the nodal basis at the gauss points by finding the derivatives of the modal function, and using

$$\mathcal{V}_{\xi,gpts} \mathcal{V}^{-1} = \mathcal{V}_{\xi,N,gpts} \tag{3.67}$$

where subscript $(.)_\xi$ indicates a derivative with respect to coordinate ξ.

Lastly, using the above method, the discrete elemental operators, such as the mass matrix $\mathbf{M}$, can be created and stored.

Two remaining issues include the choice of nodal locations and the condition number of $\mathcal{V}$. A well-behaved set of nodal locations allow accurate interpolation and a poorly-behaved set results in interpolation with large oscillations. The condition number of $\mathcal{V}$ for the modal basis can affect the accuracy when forming the discrete operators. The first issue is dealt with by optimizing a set of nodal points to minimize the Lebesque constant (see chapter 6 of Hesthaven and Warburton (2008)), and the second is resolved by using an orthogonal modal basis such as the Koornwinder basis (Hesthaven and Warburton, 2002).

Chapter 4

Biogeochemical Reaction Equation

4.1 Introduction

The focus of this chapter is accurate numerical modeling of biogeochemical processes in the ocean, to demonstrate understanding of DG FEM. For an introduction to biogeochemical modelling, the reader is referred to Fennel and Neumann (2004). This work is important in order to predict biological events such as plankton blooms. Blooms of toxic phytoplankton can be harmful to humans and marine life. Also, phytoplankton blooms are accompanied by an increased population of fish which can be harvested for human consumption. Finally, being able to predict biogeochemical ocean processes enables the study of these processes which can lead to a better understanding of the ocean ecosystem.

Biogeochemical models may contain a large number of biological or chemical constituents. The simplest models often only use Nutrient, Phytoplankton, and Zooplankton as constituents, and are commonly called NPZ models. More complicated models (Besiktepe et al., 2002) can be adaptive and contain, for example, 24 constituents. Each constituent requires the solution of advection-diffusion-reaction (ADR) equations, governed by

$$\partial_t \varphi_i + \nabla \cdot (\vec{u}\varphi_i) - \nabla \cdot \kappa \nabla \varphi_i = S_{\varphi_i}(\varphi) \tag{4.1}$$

where $\vec{u} = [u^x(x,y,t) \quad u^y(x,y,t)]$ is the velocity; κ is the diffusivity; and $\varphi = [\varphi_1, \varphi_2, ..., \varphi_n]^T$ is a vector of constituents, i referring to the i^{th} constituent, and n being the total number of constituents. Note that the source term for constituent i ($S_{\varphi_i}(\varphi)$) can be a function of any number of the constituents. The source terms describe "reactions", and often lead to chaotic dynamics.

For this work a simple NPZ model (Flierl and McGillicuddy, 2002) is used with the following source terms

$$\begin{aligned} S_N(\phi_N, \phi_P, \phi_Z) &= -\mathcal{U}e^{z/h}\frac{\phi_P\phi_N}{\phi_N + k_s} + d_P\phi_P + d_Z\phi_Z \\ &\quad +(1-a)\frac{g}{\nu}\phi_Z(1 - e^{-\nu\phi_P}) \qquad (4.2) \\ S_P(\phi_N, \phi_P, \phi_Z) &= \mathcal{U}e^{z/h}\frac{\phi_P\phi_N}{\phi_N + k_s} - d_P\phi_P - \frac{g}{\nu}\phi_Z(1 - e^{-\nu\phi_P}) \qquad (4.3) \\ S_Z(\phi_N, \phi_P, \phi_Z) &= -d_Z\phi_Z + a\frac{g}{\nu}\phi_Z(1 - e^{-\nu\phi_P}) \qquad (4.4) \end{aligned}$$

where the parameters are explained in Table 4.1, the subscripts $(.)_N, (.)_P, (.)_Z$ refer to Nutrients, Phytoplankton, and Zooplankton respectively, and lowercase z refers to the depth coordinate decreasing towards the bottom and taken as $z = 0$ at the surface. Note that three equations are not required, and the third constituent could be calculated from a conservation equation. For instance, we could calculate ϕ_N using:

$$\phi_N = \mathcal{N}_T - \phi_P - \phi_Z \qquad (4.5)$$

where $\mathcal{N}_T$ is the total biomass, which in general is a function of time and space, but is taken as a constant. In this work all three equations are solved, where the conservation equation 4.5 serves as a check of the conservativity of the numerical scheme.

This Chapter is organized such that individual components of the complete numerical solution are solved in the sections leading up to the final section, which solves the full ADR equations.

Parameter	Description	Value 1/ Value 2	[units]
$\mathcal{U}$	Phytoplankton uptake rate	0.6	[1/day]
k_s	Saturation rate of phytoplankton	0.1	[μmol/L]
d_P	Mortality rate of Phytoplankton	0.016	[1/day]
d_Z	Mortality rate of Zooplankton	0.08/0.06	[1/day]
g	Grazing rate of Zooplankton	0.1/0.13	[$L/(\mu$mol$\cdot$ day)]
a	Assimilation (efficiency) rate	0.4	[]
h	e-folding depth for light (photosynthesis)	17	[m]
ν	Parameter for Ivlev form of grazing function	0.1	[L/μmol]
$\mathcal{N}_T$	Total biomass	5	[μmol/L]

Table 4.1: NPZ equation parameter description and values

4.2 Test Problem setup

The domain of interest is shown in Figure 4-1. The depth of the domain is taken as 100 units, and the width of the domain is taken as 100 units. The bottom bathymetry contains a half-ellipse with minor axis of 20 units in x and 100 units in z centered at $(x, y) = (0, -100)$ This elliptical obstacle models an idealized sea-mount which perturbs the flow. A steady potential flow field is used, and the potential flow is calculated numerically using an IP DG method (see Section 3.3.1) with MATLAB code provided by Hesthaven and Warburton (2008). For the velocity field, slip boundary conditions are used on the top and bottom boundaries by specifying the value of the streamline at those boundaries, and periodic conditions are are used for the left and right sides of the domain. Streamline of the flow field are plotted in Figure 4-1. Specifically we take

$$\psi(x, z = 0) = 0 \tag{4.6}$$

$$\psi(x, z = bottom) = -100 \tag{4.7}$$

$$\psi(x = -50, z) = \psi(x = 50, z) \tag{4.8}$$

where ψ is the stream function, giving a flow from left to right. The solved potential flow field is scaled by factor of two, which is equivalent to multiplying ψ by two, or taking $\psi(x, z = bottom) = -200$.

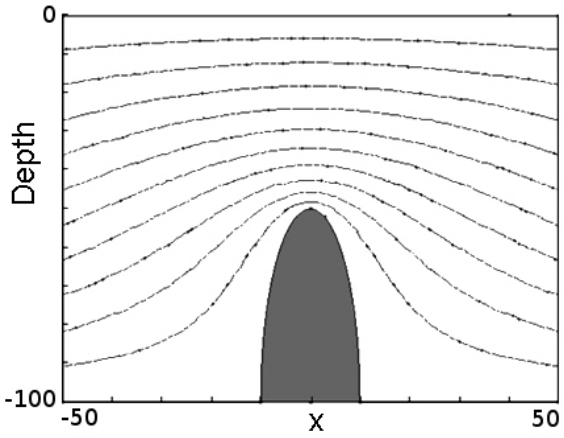

Figure 4-1: Problem domain for two-dimensional biogeochemical reaction equations, with velocity streamlines plotted.

For the biological constituents, no flux boundary conditions are specified at the top and bottom boundaries, and periodic conditions are also taken at the left and right walls. The initial concentrations are calculated from a steady-state solution for no flow, and the equations are integrated for 100 days.

For this idealized study, the diffusion terms are set to zero, that is $\kappa = 0$, which allows fully explicit methods to be used, because the numerical stability criteria does not limit the size of the computations.

4.3 One-dimensional NPZ equations

Before solving the full test problem, the NPZ equations are solved in depth to examine the behaviour of the non-linear source terms. First the steady state solution is calculated in order to initialize the numerical solution with a biologically-feasible field, and second time integration schemes are examined to ensure the solution is stable for the chosen parameter sets.

4.3.1 Steady State Solution

The steady state solutions for this model in depth can be solved for by setting the left hand side of equation 4.1 equal to zero and substituting the form of the source

terms:

$$
\begin{aligned}
0 &= -\mathcal{U}e^{z/h}\frac{\phi_P\phi_N}{\phi_N+k_s} + d_P\phi_P + d_Z\phi_Z \\
&\quad +(1-a)\frac{g}{\nu}\phi_Z(1-e^{-\nu\phi_P}) \qquad (4.9)\\
0 &= \mathcal{U}e^{z/h}\frac{\phi_P\phi_N}{\phi_N+k_s} - d_P\phi_P - \frac{g}{\nu}\phi_Z(1-e^{-\nu\phi_P}) \qquad (4.10)\\
0 &= -d_Z\phi_Z + a\frac{g}{\nu}\phi_Z(1-e^{-\nu\phi_P}) \qquad (4.11)
\end{aligned}
$$

Add 4.9 and 4.10 together to find the steady state values of ϕ_P:

$$
\begin{aligned}
0 &= -\mathcal{U}e^{z/h}\frac{\phi_P\phi_N}{\phi_N+k_s} + \mathcal{U}e^{z/h}\frac{\phi_P\phi_N}{\phi_N+k_s} + d_P\phi_P - d_P\phi_P + d_Z\phi_Z \\
&\quad +(1-a)\frac{g}{\nu}\phi_Z(1-e^{-\nu\phi_P}) - \frac{g}{\nu}\phi_Z(1-e^{-\nu\phi_P}) \\
0 &= \phi_Z\left(d_Z - a\frac{g}{\nu}(1-e^{-\nu\phi_P})\right) \\
a\frac{g}{\nu}e^{-\nu\phi_P} &= a\frac{g}{\nu} - d_Z \\
\phi_P^* &= \frac{-1}{\nu}ln\left(1-\frac{\nu d_Z}{ag}\right) \qquad (4.12)
\end{aligned}
$$

Given the biological parameters, equation 4.12 can be used to solve for the steady state values of ϕ_P^*, which is constant with depth. The $(.)^*$ notation here indicates that ϕ_P^* is not the value of ϕ_P used for initialization at all depths, because the solution ϕ_P^* is only valid till a certain depth beyond which light does not penetrate, and this depth still needs to be calculated. However, in the remainder of the derivation, ϕ_P^* is treated as a known constant. Re-arranging equation 4.10 for ϕ_Z

$$
\phi_Z^* = \underbrace{\frac{\nu\mathcal{U}e^{z/h}\phi_P}{g(1-e^{-\nu\phi_P})}}_{D(z)}\frac{\phi_N}{\phi_N+k_s} - \underbrace{\frac{\nu d_P\phi_P}{g(1-e^{-\nu\phi_P})}}_{K_{\phi_P}} \qquad (4.13)
$$

and substituting this expression into 4.5:

$$\begin{aligned}
\phi_N &= \mathcal{N}_T - D(z)\frac{\phi_N}{\phi_N + k_s} + K_{\phi_P} - \phi_P \\
\phi_N^2 + k_s\phi_N &= \mathcal{N}_T(\phi_N + k_s) - D(z)\phi_N + K_{\phi_P}(\phi_N + k_s) - \phi_P(\phi_N + k_s) \\
0 &= \phi_N^2 + \phi_N \underbrace{(k_s - \mathcal{N}_T + D(z) - K_{\phi_P} + \phi_P)}_{B(z)} + \underbrace{(-K_{\phi_P} - \mathcal{N}_T + \phi_P)k_s}_{C_{\phi_P}} \\
\phi_N^*(z) &= \frac{-B(z) \pm \sqrt{B(z)^2 - 4C_{\phi_P}}}{2} \qquad (4.14)
\end{aligned}$$

With equations 4.12 and 4.14, the initial condition can then be numerically calculated, given a numerical array of depths values $\mathbf{z}$:

$$\begin{aligned}
\vec{\phi}_Z &= \max(\mathcal{N}_t - \phi_N^*(\mathbf{z}) - \phi_P^*, 0) \qquad (4.15) \\
\vec{\phi}_P &= \phi_P^* \frac{\vec{\phi}_Z}{\max(\vec{\phi}_Z, 10^{-16})} \qquad (4.16) \\
\vec{\phi}_N &= \mathcal{N}_T - \vec{\phi}_P - \vec{\phi}_Z \qquad (4.17)
\end{aligned}$$

Using this procedure the initial steady state can be calculated numerically and used to initialize simulations. When the correct root for ϕ_N^* is chosen, a stable equilibrium is obtained, but this is not the only equilibrium of the system. Burton (2009) describes the other equilibria of this particular NPZ model, and also examines the stability of the system for various parameters.

4.3.2 Temporal convergence

It is important to examine the behaviour of the equations over time for different depths because, for certain parameter sets, the equations may become chaotic. With chaotic solutions, the numerical solution may appear to be unstable, while the scheme is correct. We found that this is particularly important for unstructured meshes where element edges are not necessarily aligned with the depth. Therefore it is worthwhile to look at the most simplified problem first, the zero-velocity, biological reactions

only, system:

$$\frac{d\phi_N}{dt} = -\mathcal{U}e^{z/h}\frac{\phi_P\phi_N}{\phi_N+k_s} + d_P\phi_P + d_Z\phi_Z + (1-a)\frac{g}{\nu}\phi_Z(1-e^{-\nu\phi_P}) \quad (4.18)$$

$$\frac{d\phi_P}{dt} = \mathcal{U}e^{z/h}\frac{\phi_P\phi_N}{\phi_N+k_s} - d_P\phi_P - \frac{g}{\nu}\phi_Z(1-e^{-\nu\phi_P}) \quad (4.19)$$

$$\frac{d\phi_Z}{dt} = -d_Z\phi_Z + a\frac{g}{\nu}\phi_Z(1-e^{-\nu\phi_P}) \quad (4.20)$$

An explicit first-order Euler scheme is compared to an explicit fourth-order low storage Runge-Kutta (LSRK) scheme (Hesthaven and Warburton, 2008, Carpenter and Kennedy, 1994) for time integration. If an implicit scheme was used, one would have to to solve the non-linear source terms using, for example, a Newton-Raphson iterative solver at each time step. Here, since we utilize an explicit time integration scheme, the size of the stable time step may be limited by the timescale of the biological reaction instead of the time step size for the advection terms in the solution of the two-dimensional ADR equations. Therefore many one-dimensional numerical tests were conducted in order to find the limiting time-step sizes and appropriate initial conditions for the biological equations. For the one-dimensional numerical tests, the following parameters were varied:

- Timestep size: [4,2,1.01,1,0.99,0.9,0.5,0.25,0.1,0.005]
- Solver: [first order explicit Euler, fourth order explicit LSRK]
- Biological Parameters: [parameter set 1, parameter set 2]
- Initial Conditions: [Constant, Steady State with parameter set 1, Steady State with parameter set 2]

Figures 4-2 and 4-3 show the final depth concentration profile of parameter set 1 and parameter set 2 respectively for the two time integration schemes after 100 days starting from the steady state initial states of parameter set 2 and parameter set 1 respectively. The true solution is taken as the final profile found using LSRK with a small time step size of 0.005 days.

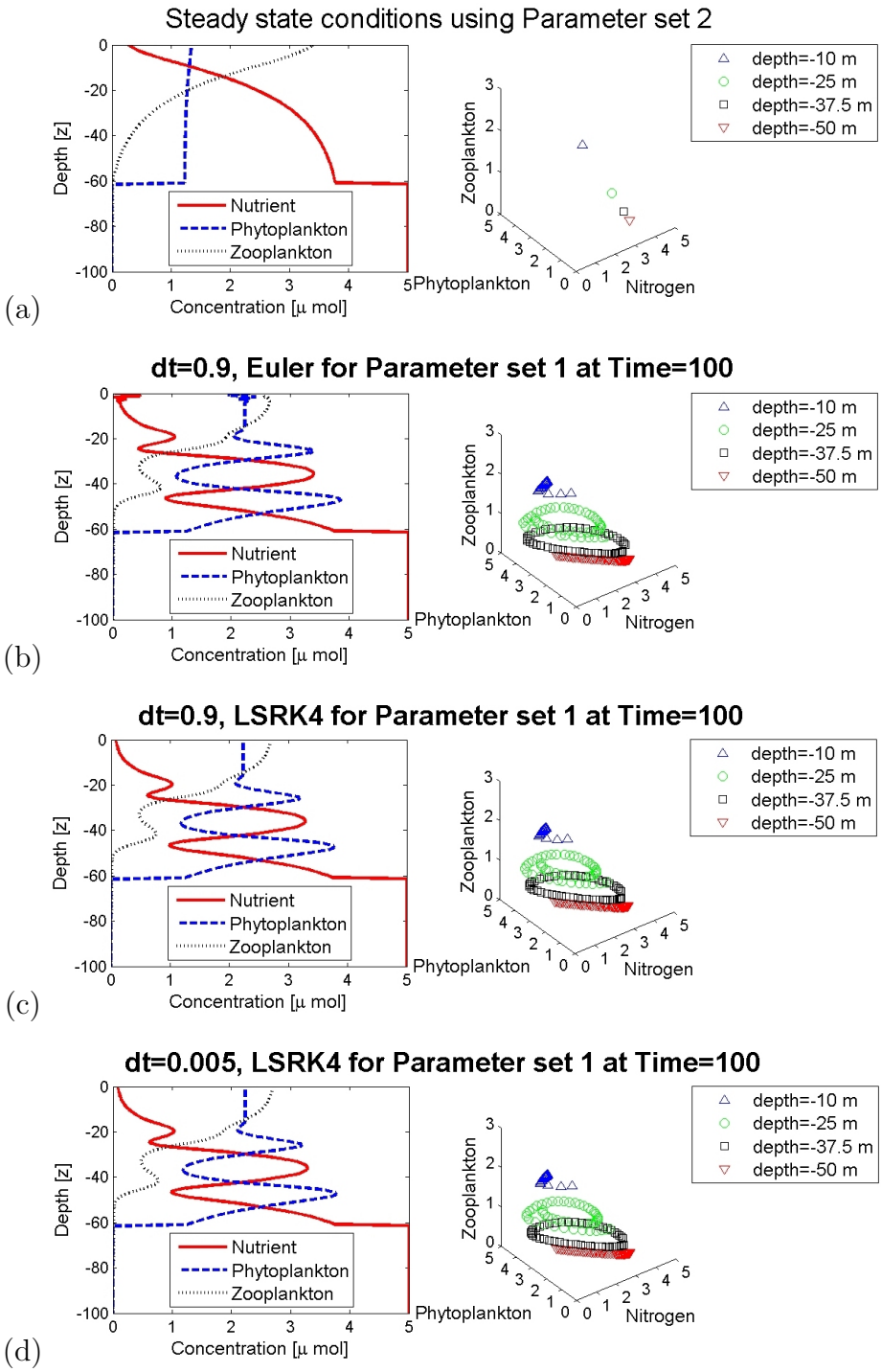

Figure 4-2: Concentration of biological constituents with parameter set 1 after 100 days of integration using the steady-state solution of parameter set 2 for the initial condition (a). The bottom plot (d) is taken as the true solution and uses a small time step with the LSRK time integration scheme. Plot (c) uses LSRK with a large time step, and the plot (b) uses the Euler time integration scheme. The plots (a-d) show the concentration in the depth of each constituent on the left, and the evolution of the amount of each constituent at different depths (or their 'orbits') on the right.

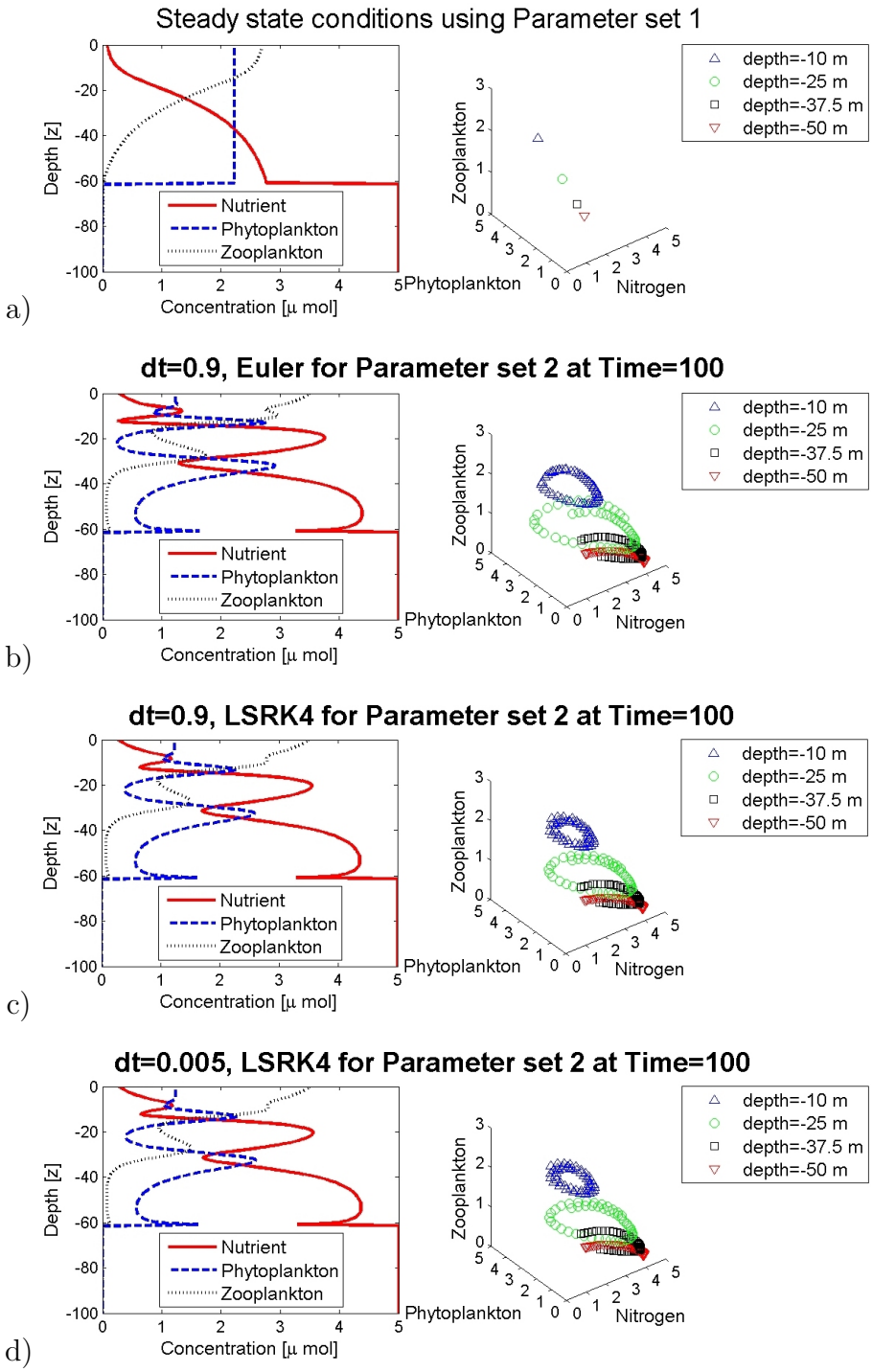

Figure 4-3: Concentration of biological constituents with parameter set 2 after 100 days of integration using the steady-state solution of parameter set 1 for the initial condition (a). The bottom plot (d) is taken as the true solution and uses a small time step with the LSRK time integration scheme. Plot (c) uses LSRK with a large time step, and the plot (b) uses the Euler time integration scheme. The plots (a-d) show the concentration in the depth of each constituent on the left, and the evolution of the amount of each constituent at different depths (or their 'orbits') on the right.

Of the two time integration schemes tested, the LSRK scheme performed the best. The Euler time integration scheme is not as accurate as the LSRK scheme, since the final profile using a time step of 0.9 days does not match the "true" solution, whereas the LSRK scheme does match the "true" solution when using the large 0.9 day time step. Both schemes become less stable/accurate at shallow depths when the time step size exceeds 1 day. While the timescale of the equations depend on the relative concentrations of the constituents and the depth, these results suggests that the timescale is on the order of 1 day. A detailed analysis of the timescales involved with this system is carried out in Burton (2009), where it was found that the biological timescales vary with depth and the value of concentration at a point. In Burton (2009), it was found that the timescales vary between zero and 0.2 days, which is faster than what we found for numerically consistent answers. With the results of Burton (2009) in mind, we use the LSRK scheme with a time step size smaller than 0.2 days. Note that for longer integration times, these results may not hold, and a smaller time step size may be required for an accurate solution. Longer integration times not only allow small errors to grow, but may put the biological system into a state which has a faster response time, leading to a smaller time scale.

There are a number of differences between the solutions for the two different parameter sets. Recall, the second parameter set has a higher Zooplankton grazing rate and lower death rate that the first parameter set. First, notice that the concentration profiles of all the constituents at the final time step is more variable in depth for the second parameter set (see Figure 4-3) than the first parameter set (see Figure 4-2). This suggests a finer spatial discretization will be necessary to accurately capture the physics for the second parameter set. Also, looking at the evolution of the constituents or the orbits of the constituents at the shallowest depths, the second parameter set evolves much faster than the first set at the shallowest depth (-10 [m]). At the deepest depths (-37.5 and -50 [m]), the second parameter set evolves slower, and at the depth of -25 [m], the evolution between the two parameter sets are similar. This suggests that the vertical behaviour between the two parameter sets will be significantly different.

As a final test, the simulations were initialized using the steady state solution calculated using equations 4.12 to 4.17. It was found that the steady state solution was maintained throughout the integration length of 100 days, and beyond, for both integration schemes. Therefore, for the integration time of interest, the time integration scheme is stable.

4.4 Two dimensional tracer advection

4.4.1 Implementation

A number of different DG implementations were written for solving the advection-diffusion equations (that is equation 4.1 with $S = 0$). The particular implementation here uses the same LSRK time integration scheme used for the source terms along with a quadrature-free DG spatial discretization in the strong form with a nodal basis. Diffusive terms can be treated explicitly or implicitly with an LDG discretization. The implicit implementation is limited to an implicit-Euler time integration, but could be easily extended. Because only small values of κ are used for this test-case, implicit time integration is not necessary for an efficient solution scheme.

Considerable efficiency was gained by using a quadrature-free scheme, and the unfiltered use of the quadrature-free scheme was possible because the differential part of the equations are linear.

4.4.2 Higher order Advection

Before solving the two-dimensional ADR equations on the test problem specified, the numerical implementation was verified. A number of test cases were used to test the convergence of the numerical implementation, and it was found that the optimal $p + 1$ rate of convergence was achieved for a p^{th} order basis function. The test cases used include tracer advection through periodic domains with constant velocity fields, tracer advection using a rotating fields, and tracer advection using a swirling velocity field (Leveque, 1996). Additionally, a set of cases were run using the NPZ Test case

		Grid 1 $N_t = 226$	Grid 2 $N_t = 856$	Grid 3 $N_t = 3438$
$p = 1$	DOF	678	2,568	10,314
$N_p = 3$	Time [s]	16	73	681
	Time/DOF	0.0236	0.0284	0.0660
$p = 2$	DOF	1,356	5,136	20,628
$N_p = 6$	Time [s]	34	218	2,551
	Time/DOF	0.0251	0.0424	0.1237
$p = 3$	DOF	2,260	8,560	34,380
$N_p = 15$	Time [s]	79	1,473	16,141
	Time/DOF	0.0350	0.1721	0.4695

Table 4.2: Simulation time for various degrees of freedom using different order basis functions. Timing reported using 3.4GHz Intel Linux nodes

described in Section 4.2 to examine the cost of using higher order advection schemes. The results are reported in Table 4.2. Note that the cost over the simulation time scales as $Cost_{time} \sim C(T)N_t(p+1)^2$ from Hesthaven and Warburton (2008), where $C(T)$ is a function dependent on the total integration time T, N_t is the number of triangles, and p is the order of the basis. From Table 4.2 a disconcerting trend is observed. Even for a similar number of degrees of freedom, the solutions using higher order basis functions are more expensive than lower order basis functions. The reason for this is two-fold: first, because an explicit integration scheme is used, the stable time-step size decreases when using higher order bases and more steps are required to finish the integration; second, higher order bases inherently are more expensive because the local matrix operators are larger, and the local operations scale as $\mathcal{O}(N_p^2)$. These results would suggest that there is no advantage to using higher order bases for calculations; however, the accuracy of the scheme also needs to be examined.

In order to illustrate the difference in accuracy for lower and higher order schemes, a cosine bell was advected twenty times through a periodic unit square domain. The final shape of the cosine bell for the lower and higher order schemes are compared in Figure 4-4. Figure 4-4 shows that the solution using the higher order basis function is more accurate for fewer degrees of freedom with a lower computational cost. That is, the result using the lower scheme looses 20% of its original height due to numerical diffusion, while the result with the higher-order scheme looks nearly identical to the

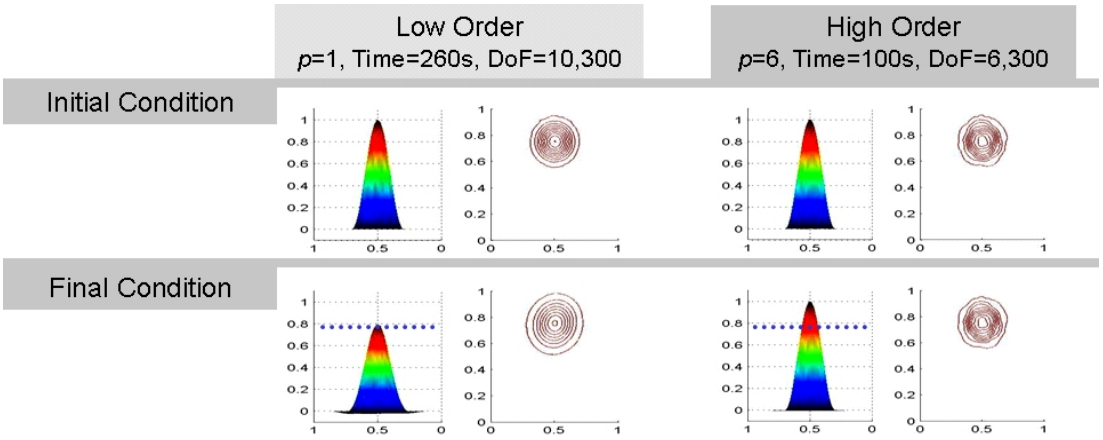

Figure 4-4: Comparison of accuracy for twenty periods of linear advection of cosine bell through periodic domain.

initial condition. These results illustrate an important point: high-order methods cannot be compared to lower order methods on a degree of freedom (DOF) basis but should be compared on an efficiency basis. The better method will have a higher accuracy at the same efficiency, or a greater efficiency at the same accuracy.

4.4.3 Test case advection with potential flow field

In order to demonstrate spatial convergence, a series of meshes are required. These are shown in Figure 4-5. The meshes were created using GMSH [1] (Geuzaine and Remacle, 2009) which is a freely available mesh generator. Note that the elements are not curved, which may lead to problems when using high order elements (Bernard, 2008).

Figure 4-6 illustrates the h and p convergence of the pure tracer advection problem for the calculated potential flow field. In Figure 4-6, G#1 (for example) refers to the use of of Grid 1 as labeled in Figure 4-5. The top row of Figure 4-6 shows the solution being refined from left (coarser grid) to right (finer grid), but retaining the same features, as the mesh is refined. The bottom row of Figure 4-6 shows the solution using a lower (left) and higher (right) order basis function, but retaining the same features as the order of the basis is increased. Qualitatively, this indicates that the

[1] www.geuz.org/gmsh

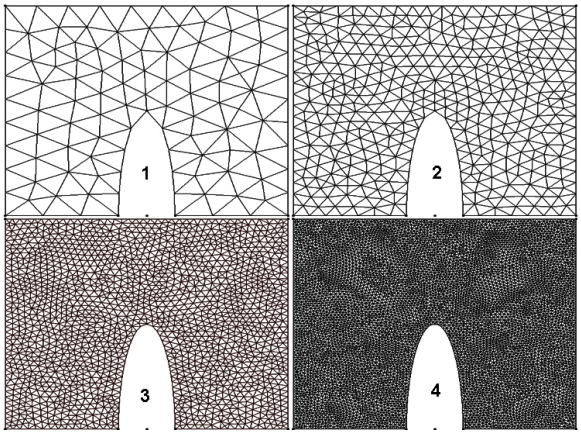

Figure 4-5: GMSH created meshes used for convergence studies.

solution is converging when refining h, the mesh, and p, the order of the basis.

Next, temporal convergence is demonstrated for the test problem. Figure 4-7 shows that the same final flow is obtained for a relatively large time step (bottom left plot) compared to a smaller time step (top left plot). Additionally, Figure 4-7 illustrates that the LSRK scheme allows a larger time step to be used than what is allowed for the Euler time integration scheme (bottom right plot), which becomes unstable for $dt = 0.07$. The top right plot shows the initial condition for this flow. Thus, qualitatively, this indicates that the solution is converging when refining the time step size, and LSRK allows a larger time step compared with the Euler scheme to be used while maintaining numerical stability,

4.5 Solution of biogeochemical reaction equations

The numerical implementation for this work was compared with the code implemented by (Burton, 2009), and the models agreed. Note that the stable time step size for advection is smaller than the stable time step for biology. Therefore the time step size of the ADR equations is limited by advection for the chosen biological parameters.

The solution at the final time is shown in Figure 4-8 for the first parameter set, and in Figure 4-9 for the second parameter set. Both simulations used third-order

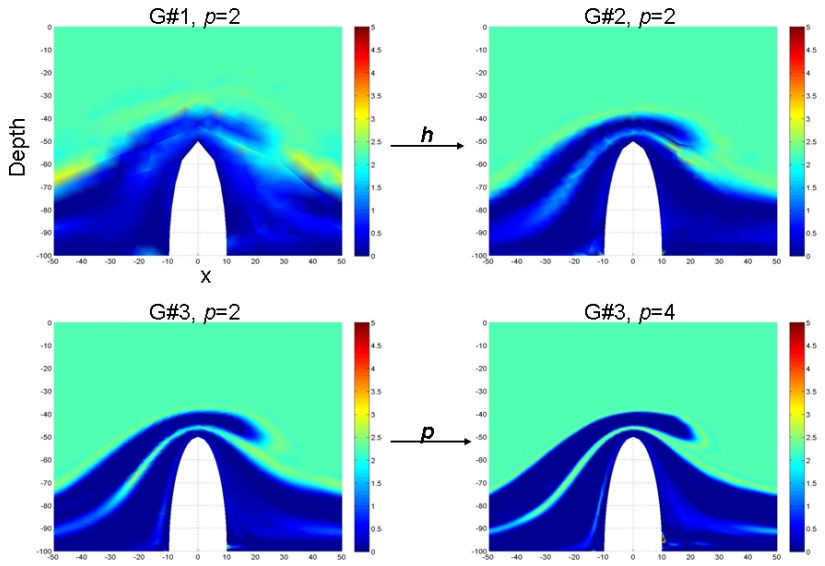

Figure 4-6: $h-p$ convergence of purely advective test case for the NPZ test case problem. G# refers to the grid use, as indicated in Figure 4-5, h refers to the size of elements, and p refers to the order of the basis. The top row demonstrates h convergence, that is the same solution is maintained as the mesh is refined. The bottom row demonstrates p convergence, that is the same solution is maintained as the order of the basis is increased.

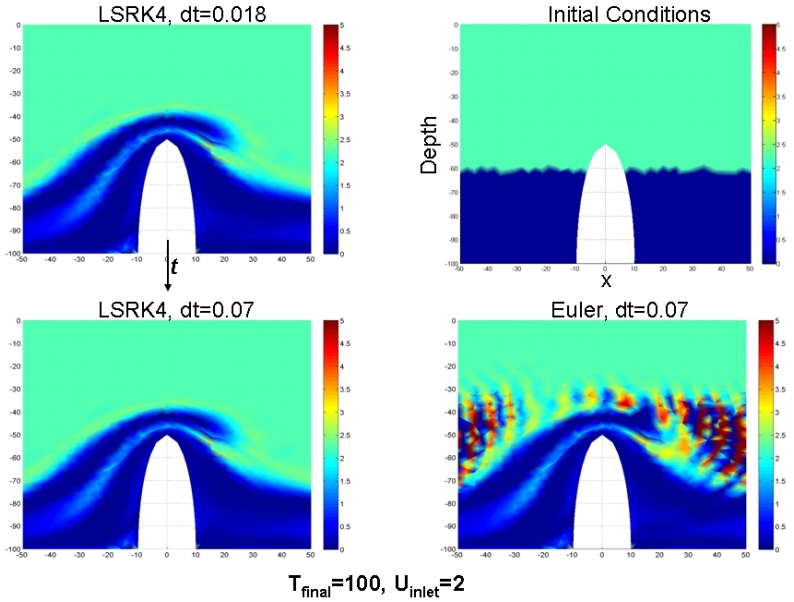

Figure 4-7: Temporal convergence of purely advected flow. The reference solution is calculated using a small (dt=0.018) time step (top left plot) and the bottom row gives the solution for larger time steps using LSRK (left) and Euler (right) time-stepping schemes. The initial conditions are plotted on the top right.

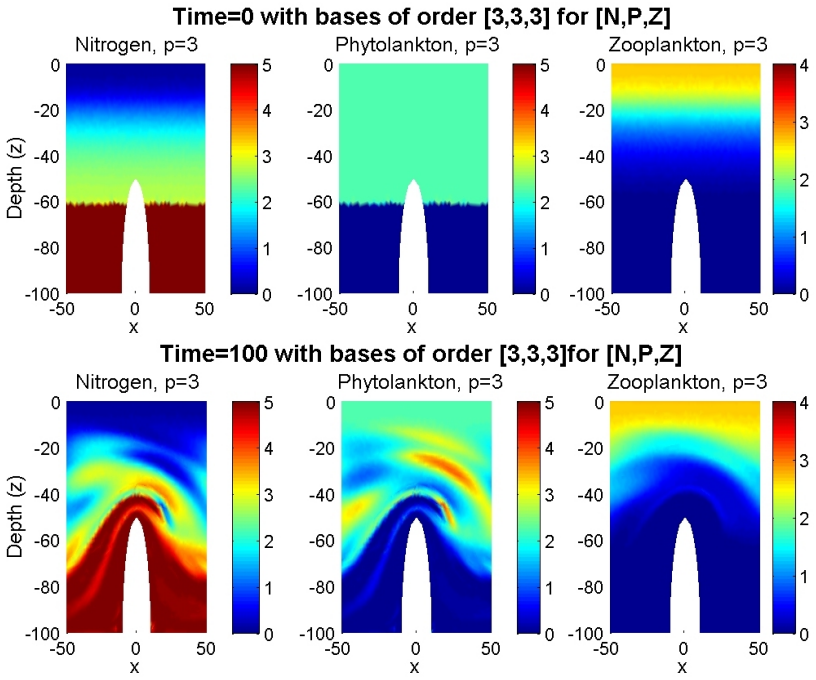

Figure 4-8: Initial and final time step for NPZ two dimensional test case using parameter set 1 on grid 2 with third-order basis functions. The simulation took 446 seconds, and color bars shown are in [μmol].

basis functions on grid 2 (see Figure 4-5).

Examining the solutions, there are a number of interesting features. Note that the flow field chosen, combined with the length of integration and the periodic boundary conditions causes the fluid to pass over the obstruction approximately two times, where the exact value depends on the specific depth. The final solution is significantly different than the initial solution which shows that the advection has a significant affect on the biological fields. Specifically, the final solution is less uniform containing finer structures than the initial solution. In particular, there is a region of significant Phytoplankton growth downstream and above the obstacle (depth around -30) that develops within the first 50 days and is maintained throughout the simulation. What the model is capturing is nutrient-rich water being brought up from below into the light-penetrating region. These nutrients are consumed by Phytoplankton resulting in a "bloom." The maintenance and stability of this bloom is unique to this parameter set and flow conditions, and cannot be expected in general.

There are also a number of differences between the two parameter sets. The second

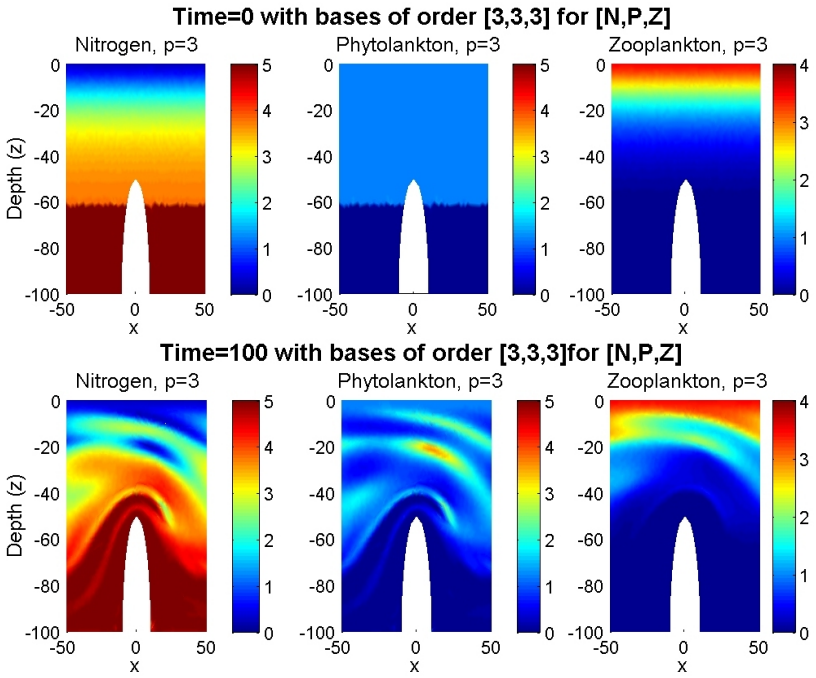

Figure 4-9: Initial and final time step for NPZ two dimensional test case using parameter set 2 on grid 2 with third-order basis functions. The simulation took 423 seconds, and color bars shown are in [μmol].

parameter set has a higher Zooplankton grazing rate (0.13 compared to 0.1), and a lower Zooplankton mortality rate (0.06 compared to 0.08), which effectively increases the amount of Zooplankton in the system. The effect of increased Zooplankton can be seen in the Zooplankton and Phytoplankton plots, where a higher concentration of Zooplankton and a lower concentration of Phytoplankton is present in Figure 4-9 than in Figure 4-8. Therefore, for the second parameter set, Phytoplankton is being consumed at a faster rate by Zooplankton. With less Phytoplankton, less Nutrients are being consumed, and more Nutrients remain in the system. A particularly interesting feature is the smoothly varying Zooplankton field for parameter set 1. Because this field is more uniform, one can conceivably gain some computational efficiency by decreasing the order of the basis used to calculate this field. This observation leads to the discussion in the next section, where different orders of basis functions are used for the different constituents.

The numerical scheme performs as expected. The current implementation takes 420-450 seconds for 2,600 integration steps of 22,710 degrees of freedom. The cost

of the solutions for higher values of κ were also examined. It was found that the explicit schemes became prohibitively expensive for relatively small values of κ (grid Peclet numbers < 100 and it was clear that an implicit solution would be necessary for efficiently dealing with diffusion.

4.5.1 Variable order basis functions

As observed from the results in the previous section, some computational efficiency could be gained by using different orders of basis functions for the different constituents on the same triangulation. What is being proposed here is a p-adaptive numerical scheme across constituents. h-adaptivity across constituents, that is, using different meshes for different constituents, is not considered. Normally adaptive schemes that change the discretization do so for all variables. Often, a problem with these schemes is finding an adaptation criterion based on one of the variables that improves the accuracy of the solution for all the variables. This adaptation criterion could also be based on multiple variables, but this results in regions of the mesh being refined for a variable that does not need refinement in that region to improve the accuracy of the simulation. The complexity increases dramatically when larger systems, such as a 24 constituent biological model, is used.

What is being proposed here is that each constituent adapts independently of the other two constituents based on constituent-dependent adaptation criteria. In order to explore this new idea, our MATLAB code used was extended to allow different constituents to use different orders of basis functions. As part of the extension, the code allows spatially varying orders of basis to be used on the same mesh. However, for these tests, only a global change to the order of basis function was made. Also, since appropriate adaptive criteria for the biological constituents have not yet been developed, the initially chosen order of basis function is maintained throughout the simulation time. Eventually, in order to gain maximum efficiency, the order of the basis function could be allowed to change dynamically, spatially, and independently for each constituent.

To examine the feasibility of the proposed scheme, the test case for this section was

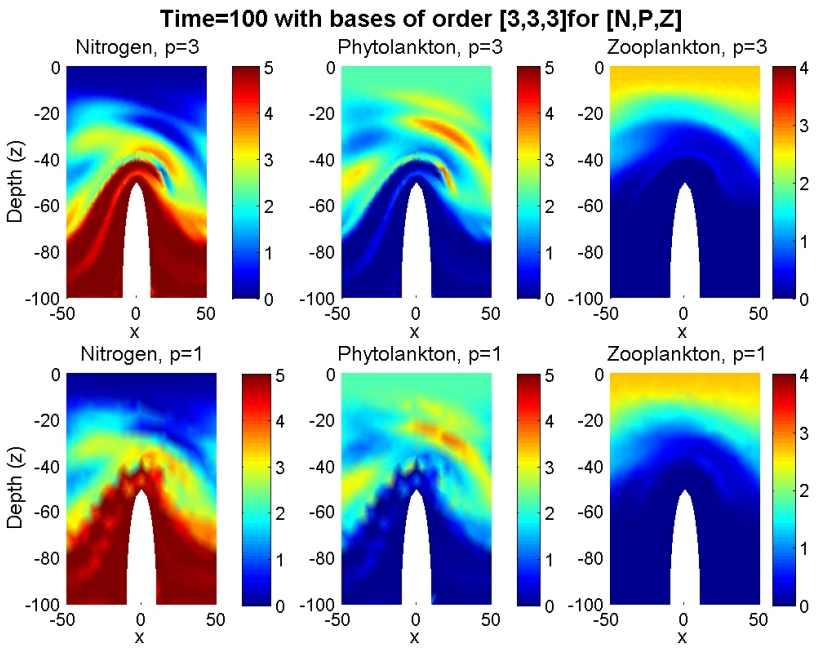

Figure 4-10: Final time step for two dimensional NPZ test case using parameter set 1 on grid 2 with third-order bases (top row) for Nitrogen, Phytoplankton and Zooplankton, taking 446 seconds. The projection of this solution onto first order bases is provided on the bottom row for comparison with the reference solution. Color bars shown are in [μmol].

run on grid 2 using third order basis functions for the Nutrient and Phytoplankton fields, and varying the order of the basis for the Zooplankton fields. The results of these simulations are plotted in Figures 4-10 to 4-12. Note that the fields are plotted both using the higher-order bases as well as the first order basis. The first-order basis fields are found by interpolating from the higher-order basis, and this allows better comparison of the results. Also, for reference, the solution using first order basis functions for all constituents is plotted in Figure 4-13. Finally, the difference between the fields in Figure 4-10 and Figure 4-11 are plotted in Figure 4-14, and the difference between the fields in Figure 4-10 and Figure 4-12 are plotted in Figure 4-15.

Qualitatively examining the solutions for the biology, the Zooplankton solutions look the same regardless of the order of basis used for this study. Also, the major features for the Nitrogen and Phytoplankton fields remain qualitatively similar. What is not obvious from the plots is the shift in the vertical position of the Phytoplankton bloom. This is highlighted by the difference plots (Figures 4-14 and 4-15) where

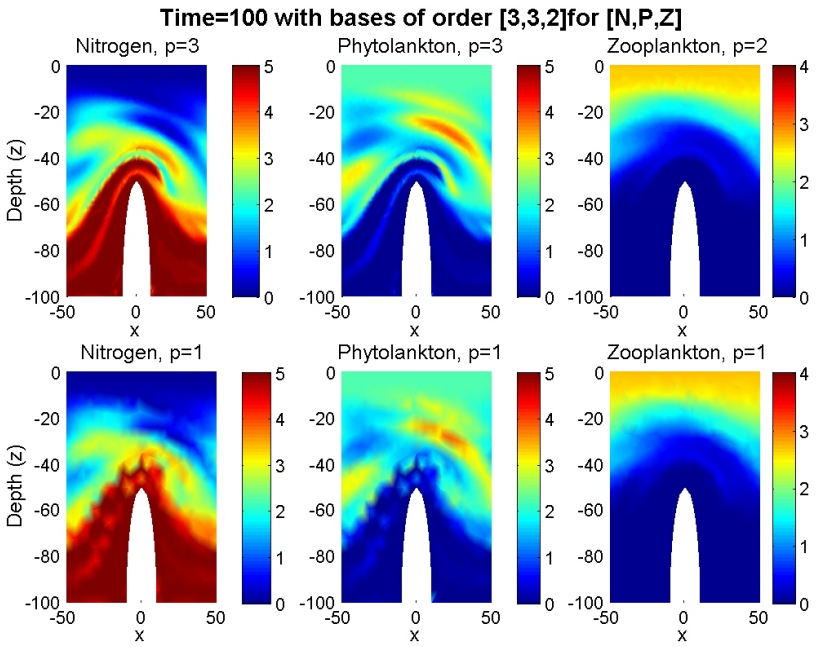

Figure 4-11: Final time step for two dimensional NPZ test case using parameter set 1 on grid 2 with third-order bases (top row) for Nitrogen and Phytoplankton, and second order basis for Zooplankton, taking 357 seconds. The projection of this solution onto first order bases is provided on the bottom row for comparison with the reference solution. Color bars shown are in [μmol].

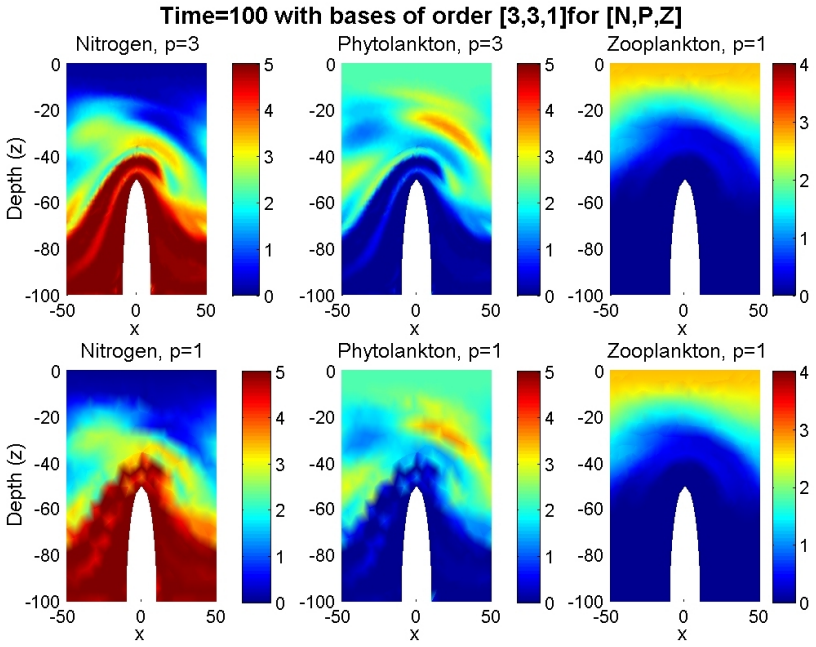

Figure 4-12: Final time step for two dimensional NPZ test case using parameter set 1 on grid 2 with third-order bases (top row) for Nitrogen and Phytoplankton, and first order basis for Zooplankton, taking 296 seconds. The projection of this solution onto first order bases is provided on the bottom row for comparison with the reference solution. Color bars shown are in [μmol].

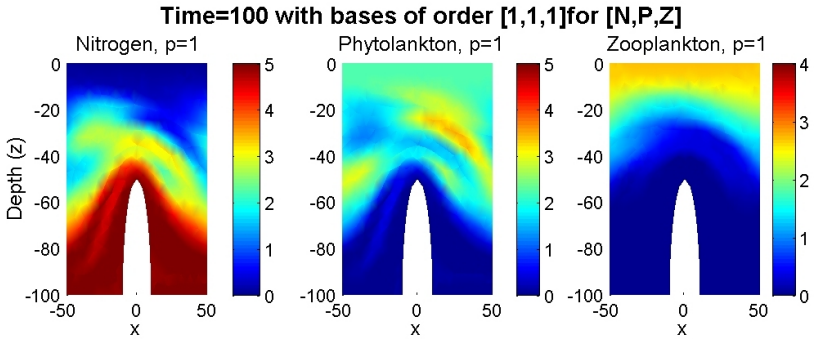

Figure 4-13: Final time step for two dimensional NPZ test case using parameter set 1 on grid 2 with first-order bases for Nitrogen, Phytoplankton and Zooplankton, taking 47 seconds. Color bars shown are in [μmol].

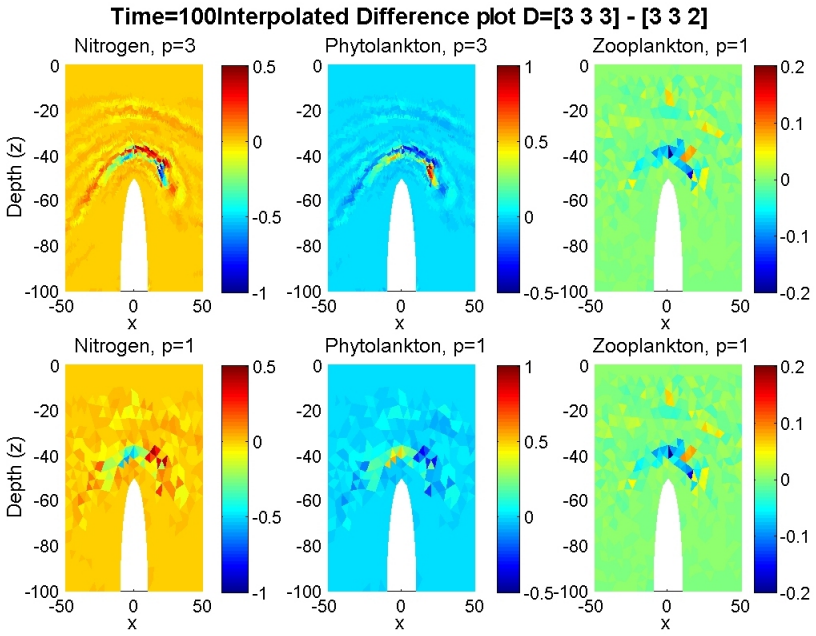

Figure 4-14: Difference between fields calculated using a third order basis for Zooplankton minus using a second order basis for Zooplankton. Note the top right plot is a projection of the difference onto a first order basis for Zooplankton. Nitrogen and Phytoplankton both use third order bases, and the difference projected onto first order bases is plotted on the bottom row. Color bars shown are in [μmol].

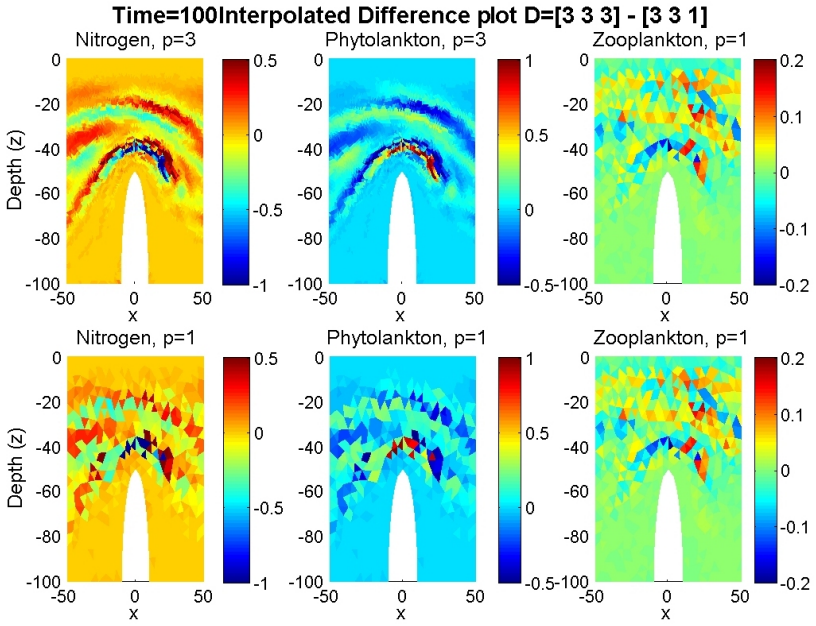

Figure 4-15: Difference between fields calculated using a third order basis for Zooplankton minus using a first order basis for Zooplankton. Nitrogen and Phytoplankton both use third order bases and the difference projected onto first order bases is plotted on the bottom row. Color bars shown are in [μmol]

the phase error causes a large difference in the solutions compared with the highest order basis function solution. Part of this error is due to the velocity field; for these simulations, the velocity field is solved on the grid that the constituent uses, which means the Zooplankton simulation uses a lower accuracy velocity field. The same phase error is present in the Nitrogen field. Apart from the phase error, the shape of the features in the Nitrogen and Phytoplankton fields are also different. In particular, a small high concentration Phytoplankton region below the main bloom is not present when the lowest order basis is used for Zooplankton. Qualitatively then, the solutions are similar enough to further examine the possibility of this type of adaptive scheme.

Examining the difference plots (Figures 4-14 and 4-15), the largest difference in the Nitrogen and Phytoplankton fields are in narrow regions. This is encouraging, because it suggests that using higher order bases for Zooplankton in the small banded regions of highest error may improve the solution. Refining the solution in the narrow bands may not improve the accuracy because it is possible that the error originates elsewhere and grows, showing up in the narrow bands. An adjoint-type refinement metric would account for the linearized part of this event. Nonetheless, the difference between the third and second order basis used for Zooplankton is smaller than the difference between the third and first order basis used for Zooplankton, as expected. Thus, refinement of the mesh somewhere will improve the accuracy of the solution, and due to the local nature of the errors, local refinement should be sufficient.

Considering the simulation time, efficiency is gained by reducing the order of the basis for Zooplankton. A 20% gain is realized by decreasing the order from three to two (calculated using $100\% \cdot (T_3 - T_2)/T_3$), wheras a 33% gain is realized by decreasing the order from three to one. This translates into significant savings, which should not necessarily be expected. Part of the efficiency of quadrature-free implementations comes from eliminating the need to interpolate. With one constituent no longer on the same grid as the other two, interpolation is required for calculating the value of the source terms. Therefore, by reducing the order of the basis, computational efficiency is gained because fewer degrees of freedom exist and matrix operators are samller, but an additional interpolation cost is introduced. If *local* p-adaption is used

within the mesh for a single consitutent, that is, various orders of bases are used for the same constituent, an additional edge interpolation is introduced for quadrature-free implementations. While not as expensive as the volumetric interpolations, this cost does effect the decision to adapt the order of the basis according to the order of neighboring elements. It is conceivable that decreasing the order of a basis in an element could increase computational cost because if the element is surrounded by neighbouring elements of equal and different order, the introduced interpolation cost could overwhelm the efficiency gain from the lower basis.

In order to examine when it would be efficient to decrease the order of a basis function, a detailed operation count for the current implementation was conducted. It was found that adapting from a high order basis to a first order basis is always more efficient. However, it was calculated that if a single element in a three-constituent model discretized uniformly with 10^{th} order basis functions is adapted down to a 9^{th} order basis, the calculations associated with that element would increase by a factor on the order of 40%. Hence, it would be considerably more expensive.

The operation count emphasized which operations are crucial for maximizing the numerical efficiency. Whether or not it is efficient to reduce the order of a basis for a constituent on a single element depends on: the number and order of basis of the other constituents on that element; and the order of the bases of the surrounding elements for the same constituent. Thus, if the adaptation criterion determines that an element for a particular constituent can be of lower order without sacrificing accuracy, the scheme needs to ensure that efficiency is also gained by the adaptation before adapting the basis.

Also, for an advanced treatment, the decision to adapt a group of constituents or a group of elements could be coupled. That is, for example, if only one constituent adapts, it may be more expensive; however if all the constituents adapt, efficiency can be gained. If the adaptation criteria only considers the cost of one constituent adapting at a time, then no adaptation may take place, whereas if the adaptation criterion considers the cost of simultaneous adaptation of constituents, adaptation may be efficient and should take place. Therefore, the most efficient adaptation

criteria would consider the cost of simultaneous constituent and element adaptation.

The cost considerations that we found thus far have applied to quadrature-free implementations, and do not apply in the same way for quadrature-based implementations. Because interpolation to gauss points is required in quadrature-based implementations, the introduced volume and edge interpolation will not add significant cost to the scheme, but will mainly contribute to the complexity of the code.

Finally, comparing the solution with uniformly third order basis functions to the solution with uniformly first order basis functions, the need for an adaptive scheme is highlighted. The small region of high Phytoplankton concentration below the main Phytoplankton bloom to the right of the bathymetry is not detectable in the simulation using only first order basis functions. While the solutions do appear similar, the fields are not as smooth as the coarser solution would suggest. Properly implemented adaptive schemes would refine the solution locally so that important small-scale features will be resolved whereas a non-adaptive lower resolution scheme would not resolve these features and they would be missed. Therefore, for improved accuracy of simulations, adaptive schemes are crucial.

4.6 Conclusions and recommendations

Temporal, h and p convergence was demonstrated for each part of the solution of the ADR equations. It was found that the LSRK integration scheme performed better than the first order Euler scheme both for integrating the biological source terms and the full ADR equations.

From the one-dimensional source terms tests, it was found that the second parameter set (higher Zooplankton grazing rate and lower Zooplankton death rate) had a finer-scale structure in the vertical direction. This finding was repeated with the full solution of the ADR equations.

The importance of comparing higher order to lower order schemes on an efficiency-accuracy basis was highlighted through a purely advective test case. The higher order scheme was more accurate, more efficient, and used fewer degrees of freedom than the

lower order scheme.

Using explicit time integration schemes with a quadrature-free method resulted in an efficient numerical scheme. However, it was found that explicit time integration of diffusive terms was prohibitively expensive for grid Peclet number smaller than 100 (or large values of κ).

Due to the uniformity of the solution for Zooplankton with the first parameter set, the use of a p-adaptive scheme (changing only the order of the basis function) across constituents was examined. It was found that such an adaptation scheme is promising for improving efficiency and accuracy of the solution, however an operation count showed that numerical cost considerations need to be made. Specifically, additional volume and edge interpolation operations result with p adaptation, and may increase the cost of a quadrature-free implementation even when the adaptation reduces the order of the basis on an element.

Finally, the need for adaptive algorithms was highlighted by noting that an important small-scale feature was unresolved in a coarse discretization, whereas a finer discretization resolved this feature. Properly implemented adaptive algorithms would locally resolve important small scale features and gain efficiency with coarse discretizations in regions where low accuracy is needed.

For the examples we considered, it is recommended that high-order quadrature-free adaptive algorithms are used whenever possible for optimum accuracy and efficiency. Also, explicit time integration is recommended for advective operators, whereas implicit time integration schemes are recommended for diffusive operators due to prohibitively expensive numerical stability constraints.

Chapter 5

Implicit Solution Techniques

The time integration of DG discretized equations is often achieved by explicit methods, such as Runge-Kutta (RK) schemes and considerable efficiency is obtained by using quadrature-free implementations. The problem with explicit time integration is that the time step size is subject to the Courant-Friedrichs-Lewy (CFL) stability condition. In a simulation with diffusion discretized using the Local Discontinuous Galerkin (LDG) method (Cockburn and Shu, 1998a), the stable time step scales as h^2/p^4 for high order basis functions (Persson and Peraire, 2006), where h is the characteristic element size and p is the order of the basis function. This stability criterion is very restrictive, and hence implicit schemes, which are not subject to the CFL stability condition, are desirable. However, implicit methods require the inversion of a large matrix, which may not be feasible to store for larger problems. Additionally, to solve the Incompressible Navier Stokes equations using a Projection Method, the inversion of a large matrix is also required. This section focuses on finding an appropriate iterative solver and preconditioner combination to solve linear advection-diffusion equations implicitly, with an emphasis on the preconditioner, but this work also enables the solution of more complicated equations such as the Incompressible Navier Stokes equations.

5.1 Review of solvers utilized for DG Schemes

DG schemes are often integrated in time using explicit schemes, and in a series of papers, Cockburn et al. (1990) introduced and analyzed their explicit Runge-Kutta DG (RKDG) scheme. The RKDG scheme has been used with considerable success. Since then, Strong Stability Preserving (SSP) RK schemes have been developed (Gottlieb et al., 2001). Explicit schemes make use of efficient matrix–vector multiplications, and are often used in practice. However, for large three-dimensional problems with widely ranging space and time scales, it is computationally efficient to use implicit time integration schemes when the CFL condition is too stringent. Some mixed implicit/explicit strategies have been suggested (Ascher et al., 1997, Kennedy and Carpenter, 2003), but the remainder of this chapter focuses on implicit schemes.

For high computational efficiency and lower memory usage, an efficient data-structure is necessary. The compressed column format (Barrett et al., 1994) should be avoided in favor of a dense block format which can minimize cache misses (Persson and Peraire, 2006). Persson and Peraire (2006) store the block diagonal and off-block-diagonal entries in separate arrays. A more sophisticated storage strategy is required for the LDG discretization of the diffusive terms, since the final matrix has additional non-zero entries. The more sophisticated strategy involves storage of a number of smaller matrices. However, to further circumvent the storage problem associated with LDG, Peraire and Persson (2007) proposed the Compact Discontinuous Galerkin (CDG) discretization.

A proper preconditioner and solver is neccessary for efficient matrix inversion. In Persson and Peraire (2006), the equations are discretized using LDG and solved using the Quasi-Minimum Residual (QMR) method, the Conjugate Gradient Squared (CGS) method, the Generalized Minimal RESidual (GMRES) method, and restarted GMRES(m) with restart value m. A $p1$–ILU(0) preconditioner was proposed and tested. This preconditioner consists of a block ILU(0) preconditioner used as a pre–smoother for a two–level p–Multi-Grid (MG) scheme. The p–MG scheme works by using an orthogonal Koornwinder expansion to project the residual from a higher-

order basis function solution to a $p = 1$ solution (Persson and Peraire, 2006). The correction is then calculated using a sparse-direct solver on the reduced problem (Persson and Peraire, 2006). Three simplified test problems solving the compressible Navier–Stokes equations were studied, and it was found that restarted GMRES(m) with $m = 20$ worked well, especially when preconditioned with their proposed $p1$–ILU(0) preconditioner (Persson and Peraire, 2006). Other researchers have also had success with p–MG schemes (Fidkowski et al., 2005).

Later, Persson and Peraire (2008) solved generalized conservation laws using a CDG discretization for the diffusive terms. In this paper, Persson and Peraire consider several preconditioner options: Block Jacobi, block Gauss-Seidel (GS), and block incomplete LU factorizations with zero fill-in ILU(0). Several other efforts are also reviewed in Persson and Peraire (2008), most making use of MG methods. It was found that the solution of pure-advection problems using an implicit scheme was more robust than the solution of implicit advection-diffusion problems. The diffusive terms were not adequately handled by an ILU(0)preconditioner, and it was shown that diffusive problems often required a MG coarse-grid correction to improve convergence and robustness (Persson and Peraire, 2008). Persson and Peraire (2008) found that the $p1$–ILU(0) preconditioner outperformed all the other preconditioner options, showing remarkable consistency and robustness over a range of test cases. GMRES was found to be the fastest and most reliable solver, in general, at the cost of increased computations and storage as the number of iterations increase Persson and Peraire (2008). The combination of the $p1$–ILU(0) preconditioner with the GMRES solver was found to be optimum in their case.

5.2 Novel studies on Solvers and Preconditioners for DG schemes

5.2.1 Description of solvers

The GMRES(m) and QMR solvers are a part of MATLAB, while the BiCGSTAB(l) algorithm was obtained from Sleijpen (2009). Each have the following syntax

```
[X,FLAG,RELRES,ITER,RESVEC] = GMRES(A,B,RESTART,TOL,MAXIT,M1,M2,X0)
[X,FLAG,RELRES,ITER,RESVEC] = QMR(A,B,TOL,MAXIT,M1,M2,X0)
[X,RESVEC,ITER*] = CGSTAB(A,B,X0,TR0,OPTIONS,M1,M2)
```

where `X` is the solution; `FLAG` contains error information; `RELRES` gives the relative error at the final iteration; `ITER and ITER*`[1] give the number of iterations; `RESVEC` is a vector of convergence history; `A` is the problem matrix; `B` is the right-hand-side vector; `RESTART` is the number m for GMRES(m); `TOL` is the convergence criteria tolerance; `MAXIT` is the maximum number of iterations; `M1` and `M2` are the preconditioner matrices [2]; and `X0` is the initial guess vector. The BiCGSTAB(l) implementation has a slightly different, but similar syntax, the major difference being that the tolerance, maximum iterations, value for l, and other options are passed to the function via the `OPTIONS` structure. Also, a helper function was required to input function handles instead of matrices for the matrix-free preconditioners.

GMRES(m)

The Generalized Minimum RESidual with restarts after (m) iterations algorithm is based on the minimization of the residual $r_n = b - \mathbf{A}x_n$ where $x_n \in \mathbf{K}_n$ and $\mathbf{K}_n$ is the Krylov subspace formed after n steps of Arnoldi iteration (Trefethen and Bau,

[1] The original function from Sleijpen (2009) had to be modified to output this outer iteration count

[2] if `M2` is empty, it is not an `LU`-type preconditioner

1997). The norm $\|\tilde{\mathbf{H}}_n y - \|b\|e_1\|$ is minimized for y, and then $x_n = \mathbf{Q}_n y$ (Trefethen and Bau, 1997), where $\mathbf{A}\mathbf{Q}_n = \mathbf{Q}_{n+1}\tilde{\mathbf{H}}_n$ and $e_1 = (1, 0, ..., 0)$. That is, the columns of $\mathbf{Q}_n$ are the first n columns of $\mathbf{Q}$ in the $\mathbf{QR}$ factorization of $\mathbf{A}$, and $\tilde{\mathbf{H}}_n$ is the upper Hessenberg matrix obtained at the n^{th} Arnoldi iteration. This algorithm works to solve the problem because the ever-increasing size of the Krylov subspace approaches the span of the columns of $\mathbf{A}$. Minimizing the residual in the subspace of $\mathbf{A}$ is the same as solving the problem exactly, and will happen when $n = rank(\mathbf{A})$, if $\mathbf{A}$ has full rank. The solution procedure orders $\mathbf{A}$ according to its dominant eigenvalues, hence, its dominant reduced-rank (generalized inverse) component.

GMRES becomes more expensive both in terms of memory and computation as n becomes large, hence restarted GMRES(m) is often used, where the algorithm is restarted using x_m as the initial guess after m iterations. Where the convergence of GMRES is monotonic with increasing iterations (Trefethen and Bau, 1997), GMRES(m) may stagnate (Persson and Peraire, 2008). Each iteration of GMRES(m) requires one matrix-vector multiplication, but becomes more expensive as m increases.

QMR

The Quasi-Minimum Residual method is also a Krylov subspace minimum residual method, but the Krylov subspace is different in this case. QMR is based on tridiagonal biorthogonalization methods. Here the $\mathbf{A}$ matrix is factored into $\mathbf{A} = \mathbf{V}\mathbf{T}\mathbf{V}^{-1}$, where $\mathbf{T}$ is tri-diagonal, and $\mathbf{V}$ is non-singular (but not unitary) and the columns of $\mathbf{V}$ are orthogonal to the columns of $\mathbf{W} = (\mathbf{V}^{-1})^*$. The Lanczos-like equations that follow are

$$\mathbf{A}\mathbf{V}_n = \mathbf{V}_{n+1}\tilde{\mathbf{T}}_n \tag{5.1}$$

$$\mathbf{A}^*\mathbf{W}_n = \mathbf{W}_{n+1}\tilde{\mathbf{S}}_n \tag{5.2}$$

$$\mathbf{T}_n = \mathbf{S}_n^* = \mathbf{W}_n^*\mathbf{A}\mathbf{V}_n \tag{5.3}$$

where $\mathbf{V}_n, \mathbf{W}_n$ are $m \times n$, $\tilde{\mathbf{T}}_n$ and $\tilde{\mathbf{S}}_n$ are $(n+1) \times n$ non-hermitian tri-diagonal matrices, and $\mathbf{T}_n$ and $\mathbf{S}_n$ are the upper $n \times n$ blocks of $\tilde{\mathbf{T}}_n$ and $\tilde{\mathbf{S}}_n$ respectively

(Trefethen and Bau, 1997). From these equations a three-term recurrence relation results:

$$\mathbf{A}v_n = \tilde{\mathbf{T}}_{n-1,n}v_{n-1} + \tilde{\mathbf{T}}_{n,n}v_n + \tilde{\mathbf{T}}_{n+1,n}v_{n+1} \tag{5.4}$$

$$\mathbf{A}^* w_n = \tilde{\mathbf{S}}_{n-1,n}w_{n-1} + \tilde{\mathbf{S}}_{n,n}w_n + \tilde{\mathbf{S}}_{n+1,n}w_{n+1} \tag{5.5}$$

Therefore, starting with arbitrary vectors V_1, w_1 such that $v_1^* w_1 = 1$, and setting $\tilde{\mathbf{T}}_{1,2} = \tilde{\mathbf{T}}_{2,1} = 0$, and $v_0 = w_0 = 0$. Then, for each $n = 1, 2, ...$ set $\tilde{\mathbf{T}}_{n,n} = w_n^* A v_n$, determine v_{n+1}, w_{n+1} from equations 5.4-5.5 (up to a constant), then find $\tilde{\mathbf{T}}_{n+1,n-1}, \tilde{\mathbf{T}}_{n+1,n+1}$ subject to the normalization of $w_{n+1}^* v_{n+1}$ again using 5.4-5.5. Consequently, the subspaces $v_n \in \langle v_1, \mathbf{A}v_1, ..., \mathbf{A}^{n-1}v_1 \rangle$ and $w_n \in \langle w_1, \mathbf{A}^* w_1, ..., (\mathbf{A}^*)^{n-1} w_1 \rangle$ are formed (Trefethen and Bau, 1997). QMR chooses the normalized initial residual $(b - \mathbf{A}x_0)$ as the initial $\mathbf{V}_1$, and also introduces a weighted scaling matrix (Freund and Nachtigal, 1991). This algorithm uses a look-ahead Lanczos iteration in order to avoid *near-breakdowns* [3] (Freund and Nachtigal, 1991). A disadvantage of this method is that it needs the computation of $\mathbf{A}^*$ (5.2), and subsequently requires also $\mathbf{M}^*$ for preconditioner $\mathbf{M}$. Each iteration of QMR requires two matrix-vector multiplications.

BiCGSTAB(l)

The BiConjugate Gradient STABilized uses an l-degree minimum residual (MR) polynomial (Sleijpen and Fokkema, 1993) and is yet another Krylov subspace minimum residual method. This is a variant of the BiConjugate Gradient (BCG) method (Trefethen and Bau, 1997). In BCG, the initial $\mathbf{V}_1$ is chosen as b. This has the effect of choosing x_n in the same subspace as GMRES (i.e. $\langle b, \mathbf{A}b, ..., \mathbf{A}^{n-1}b \rangle$), but minimizing the residual in the $\langle w_1, \mathbf{A}^* w_1, ..., (\mathbf{A}^*)^{n-1} w_1 \rangle$ subspace (Trefethen and Bau, 1997). The BiCGSTAB(l) algorithm tries to both smooth the convergence rate of BCG, and account for the *near-breakdown* situations (Trefethen and Bau, 1997). The BiCGSTAB(l) algorithm has one inner loop that consists of two parts. In the first part, termed the "BCG part," new BCG vectors are computed implicitly by com-

[3] Near-breakdown happen when $w_{n+1}^{\mathbf{T}} v_{n+1} \approx 0, w_{n+1} \approx 0, v_{n+1} \approx 0$

puting the iteration coefficients α and β explicitly (Sleijpen and Fokkema, 1993). In the second part, termed the "MR part," a locally minimum residual is calculated using the Minimum Residual approach (Sleijpen and Fokkema, 1993). At the end of the inner loop, the residuals and BCG vectors (including the current solution) is updated (Sleijpen and Fokkema, 1993). Each outer step of BiCGSTAB(l) requires $2l$ matrix-vector multiplications, but becomes more expensive for higher l (Sleijpen and Fokkema, 1993). More information can be obtained from Sleijpen (2009) and Sleijpen and Fokkema (1993).

5.2.2 Description of DG-specific Preconditioners

A preconditioner, $\mathbf{M}$, is a matrix that approximates $\mathbf{A}$ and is easy to invert. Left preconditioning involves left multiplication of $\mathbf{M}^{-1}$ to the original linear system. That is, $\mathbf{A}x = b$ becomes $\mathbf{M}^{-1}\mathbf{A}x = \mathbf{M}^{-1}b$. For $\mathbf{M} = \mathbf{A}$, $\mathbf{M}^{-1}\mathbf{A}x = \mathbf{I}x = x = \mathbf{A}^{-1}b$, the problem is solved exactly, and for $\mathbf{M} = \mathbf{I}$, the original problem remains.

Block Jacobi

One of the most important preconditioners used in practice is the Jacobi preconditioner (Trefethen and Bau, 1997). Here the preconditioner $\mathbf{M}$ is taken as the diagonal entries of the $\mathbf{A}$ matrix. This is easily inverted, and for some problems it can give significant improvements in computational time. For DG-discretized systems, a block-Jacobi preconditioner may be used, where $\mathbf{M}$ is taken as the $N_p \times N_p$ blocks on the diagonal of $\mathbf{A}$. It is more expensive to invert this $\mathbf{M}$, but can be done on an individual block-by-block basis. The expense of this inversion becomes non-trivial when using higher-order basis functions. In Persson and Peraire (2008), the cost for computing a Jacobi factorization (including the cost of computing the matrix $\mathbf{A}$) is given as $(2/3)N_p^3 N_t$ where N_t is the number of elements in the discretization, and N_p is the number of degrees of freedom in an element. The cost of $\mathbf{M}^{-1}x$ is also given in Persson and Peraire (2008) as $(2)N_p^2 N_t$.

Block Gauss-Seidel

The Block Gauss-Seidel (GS) preconditioner is similar to the Block Jacobi preconditioner, but keeps both the diagonal blocks and all the lower (or upper) blocks. For some specific problems, usually pure advection, the GS preconditioner has been shown to perform significantly better than the Jacobi preconditioner, but for general problems only a marginal improvement can be expected (Persson and Peraire, 2008). The inversion of this matrix is difficult to obtain directly, but by using a block back-solve, the effect of the inverse can be obtained. The cost for computing the preconditioner is again given from Persson and Peraire (2008) as $(2/3)N_p^3 N_t$, and the cost of $\mathbf{M}^{-1}x$ is given as $(D+3)N_p^2 N_t$, where D is the dimension of the problem.

ILU(0)

In general, when computing the **LU** factorization of a sparse matrix, the sparsity of the matrix is lost in the computed **L** and **U** matrices. That is, **L** and **U** are more dense and require additional storage. For large systems of equations, this additional storage is prohibitive, not to mention the cost of essentially directly computing the inverse of **A**. The incomplete **LU** factorization with zero fill-in calculates the **LU** factorization of the matrix **A**, but does not allow the newly computed **L** and **U** matrices to have a sparsity pattern different from that of **A**. Not only does this reduce the storage requirements, but also the computational cost. Also, with a DG discretization, making additional assumptions about the mesh can further reduce the computational cost of the ILU(0) factorization (Persson and Peraire, 2008). The cost for computing the ILU(0) preconditioner is greater than that of the Jacobi and GS preconditioners, and is given from Persson and Peraire (2008) as $(2D+8/3)N_p^3 N_t$. The computation of $\mathbf{M}^{-1}x$ is also given in Persson and Peraire (2008) as $(2D+4)N_p^2 N_t$, which is the same as calculating the matrix-vector product $\mathbf{A}x$.

p-Multi-Grid (MG)

MG preconditioners are typically good at handling low-frequency components of the original problem, leaving the high-frequencies to be solved by other means (Trefethen and Bau, 1997). MG preconditioners are usually calculated by solving a fine-grid problem on a coarser grid, and then transferring the coarse-grid solution back onto the fine-grid (Trefethen and Bau, 1997). This transferral is trivial for structured-grid discretizations, but for unstructured grids, the transferral of the solution is less straight-forward. Fortunately, for DG discretizations with high-order basis functions, an alternative exists. Instead of projecting the solution onto a coarse grid, the solution can be projected onto a lower-order basis function. This method is used successfully by a number of researchers, including Persson and Peraire (2008). The cost for this preconditioner is not given in Persson and Peraire (2008), but depends on the order of basis function (p), the number of elements N_t, and the solution method (usually sparse-direct) used for the coarse grid.

5.3 Results

Equation 3.30 is discretized using a nodal (DG) FEM scheme with appropriate boundary conditions as discussed in Chapter 3. The exact problem being solved is discussed in detail in Chapter 4. The resultant discretized system of equations gives rise to a block matrix structure, each block being associated with a single element. Due to the definition of the fluxes $\hat{\mathbf{F}}_{inv}$ and $\hat{\mathbf{F}}_{vis}$, there are also off-block-diagonal entries in the matrix. The number of off-diagonal entries will depend on the type of discretization used for the viscous terms, where an LDG discretization will lead to a maximum of nine off-diagonal block entries, a CDG or IP disretization will lead to a maximum of three off-diagonal block entries, and an HDG discretization will lead to a smaller system with smaller blocks and a maximum of four off-diagonal block entries. In all cases, for a large number of elements, the final matrix will be reasonably sparse. An example of the sparsity pattern for both an 8–element and 104–element discretization on a structured triangular grid using the LDG fluxes is shown in Figure

5-1.

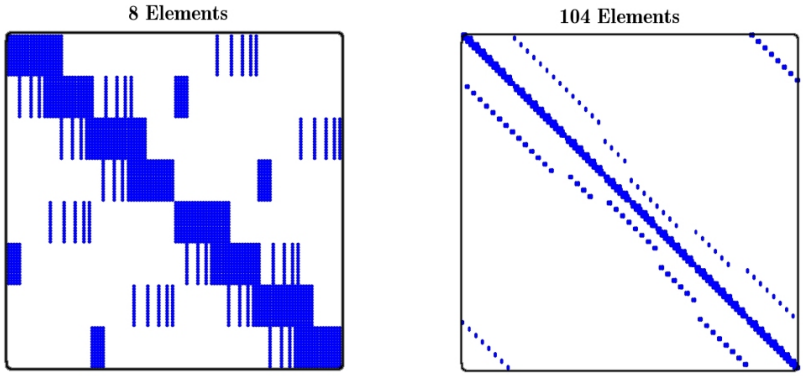

Figure 5-1: Matrix sparsity patterns for fourth order ($p = 4$) basis functions with 8 (left), and 104 (right) elements

The size of the blocks depend on the order, p, of the basis functions, the type of element (triangular or quadrilateral), and the dimension of the problem. The number of unknowns N_p in a two-dimensional triangular element scales as $N_p = \frac{1}{2}[(p+1)(p+2)]$, while the number of unknowns on each edge scales as $N_{fp} = (p+1)$ for one-dimensional edge elements. As an example, different order triangular elements are plotted in Figure 5-2.

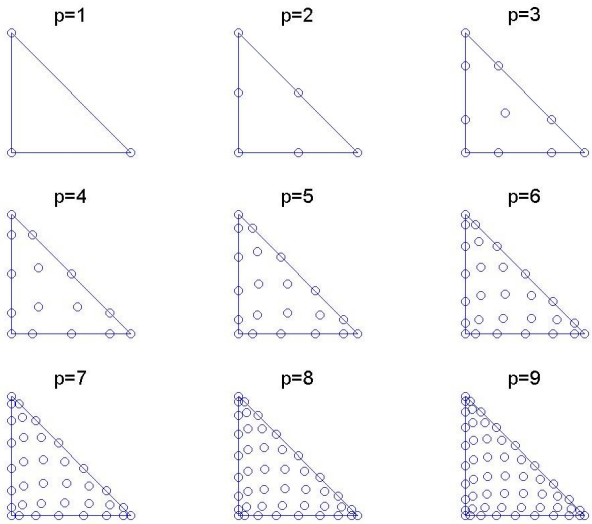

Figure 5-2: Location of unknowns on master triangle for various order (p) of basis functions

The MATLAB implementations are briefly discussed below. As a cautionary note, the efficiency of the various schemes do not reflect a realistic, optimized implementation. Some effort was expended in order to compare the cost of different preconditioners, but the main criteria for preconditioner selection should be the iteration count.

5.3.1 Constructing the A Matrix

An implementation for explicit time integration can be used directly by iterative matrix solvers because the explicit implementation essentially provides an $\mathbf{A}x$ matrix-vector multiplication. Nonetheless, for the purposes of this study it was convenient to have the matrix available for forming the various preconditioners. Also, with the matrix available, various properties of the matrix, including the sparsity pattern and eigenvalues could be easily examined. Thus, $\mathbf{A}$ is formed in all cases for this study. However, in practice a DG-specific matrix-free implementation is desirable, since it is possible to take advantage of the structure of the DG-specific matrix to improve efficiency and reduce storage.

5.3.2 Preconditioners

A number of preconditioners were tested. Most were implemented ourselves, while the MATLAB implementation of the ILU(0) preconditioner was used. The following preconditioners were tested:

1. None: $\mathbf{M} = \mathbf{I}$.

2. Upper: `M=triu(A)`, this is the non-block version of the Block GS preconditioner, and uses the upper triangle of $\mathbf{A}$

3. Lower: `M=tril(A)`, this is the non-block version of the Block GS preconditioner and uses the lower triangle of $\mathbf{A}$

4. Jacobi: `M=diag(diag(A))`, this is the classical Jacobi preconditioner

5. ILU(0): `[M1 M2]=ilu(A,setup.type='nofill')`, this is the ILU(0) implementation in MATLAB

6. Block ILU(0): This is essentially the same as the Block Jacobi preconditioner, but the LU factorization is used to compare the computation time of various implementations.

7. Block Jacobi: This preconditioner is described in §5.2.2. Two implementations are used, and are described in below.

8. Block GS: This preconditioner is described in §5.2.2. Three implementations of this preconditioner were used, and are described below.

9. MG: This preconditioner is described in §5.2.2, and the implementation is discussed below.

5.3.3 Block Jacobi Preconditioner

Essentially three implementations of this preconditioner was used, and it was expected that the same number of iterations would result for the different implementations, while the computational time would differ. This enabled the comparison of computation time for the different implementations. Since neither an efficient Block GS nor p-MG implementation existed in MATLAB, having a similarly-implemented Block Jacobi algorithm allows the comparison between preconditioners based on the number of iterations with the ability to extrapolate that result to the computational time of an efficient implementation of the algorithm.

The first two implementations were expected to have similar computational time. In the first "BlockILU" implementation, the **LU** factorization of the block diagonals were computed, and supplied to the solver functions as $\mathbf{M}1$ and $\mathbf{M}2$ respectively (see §5.2.1 for the solver syntax, and the meaning of $\mathbf{M}1$, $\mathbf{M}2$). The second "BlockJacobi" implementation simply supplied the Block Jacobi matrix as $\mathbf{M}^{-1}x$ left empty. The third "BlockJacobi2" implementation passed a "function handle" instead of a matrix

to the solver functions. The function passed in computes the product $\mathbf{M}^{-1}x$ and returns x. For this implementation, $\mathbf{M}^{-1}$ was pre-computed block-by-block and passed to the function. Hence, the only computational expense was due to the matrix-vector multiplication.

5.3.4 Block Gauss-Seidel Preconditioner

The first MATLAB implementation of the Block GS preconditioner is similar to the second implementation of the Block Jacobi preconditioner, that is, $\mathbf{M}1$ is supplied to the solver with $\mathbf{M}2$ left blank. Here $\mathbf{M}1$ contains the lower triangular blocks of $\mathbf{A}$ (that is, the lower triangle of $\mathbf{A}$ including the the upper portions of the block-diagonal entries). This implementation was found to run prohibitively slowly, but was useful for debugging the other implementations. The main difficulty with this preconditioner was obtaining an efficient implementation for testing purposes. A number of implementation were attempted, and these are described below.

The first attempt at a more efficient implementation was similar to the third Block Jacobi implementation, that is a function that performs the $\mathbf{M}1^{-1}x$ was passed to the solver. Here, the inverse of the blocks on the diagonal are precomputed and stored in $\mathbf{M}1$, and the MATLAB code is as follows:

```
1   x(1:Np)=M1(1:Np,1:Np)*x(1:Np);
2   for k=2:Nt
3      range=(k-1)*Np+1:k*Np
4      x(range)=M1(range,1:Np)*( x(range)...
5                    +A(range,1:(k-1)*Np)*x(1:(k-1)*Np) );
6   end
```

Unfortunately, this implementation took even longer to run than the original. After writing a benchmarking script it was identified that line 5 of the above algorithm

was responsible. This operation consisted of the multiplication of the sparse lower-diagonal blocks with the newly-solved-for vector. Even though the matrix was small, it was a *sparse-matrix* multiply, which has some associated overhead in MATLAB causing the computation to slow considerably. Coding this algorithm in C and using it in MATLAB through the mex interface also did not prove helpful. Since a sparse multiplication was not used in the C implementation, this code still ran prohibitively slowly. To partly overcome the overhead problem in the MATLAB implementation an `if` statement was included to only do the multiplication when necessary. The modifications to the algorithm are as follows:

```
1   x(1:Np)=M1(1:Np,1:Np)*x(1:Np);
2   for k=2:Nt
3      range=(k-1)*Np+1:k*Np
3.1    if nnz(A(range(1),1:(k-1)*Np)
4         x(range)=M1(range,1:Np)*( x(range)...
5                        +A(range,1:(k-1)*Np)*x(1:(k-1)*Np) );
5.1    else
5.2        x(range)=M1(range,1:Np)*x(range);
5.3    end
6   end
```

Unfortunately, the overhead of the `if` statement and the `nnz` function was nearly the same as the overhead of the sparse-multiply, so there were only marginal savings. This implementation was tested but never used since it still ran prohibitively slowly. In the next attempt, instead of using the sparse entries of $\mathbf{A}$, the off block diagonal entries were reconstructed using the DG operators, and is as follows:

```
1   x(1:Np)=M1(1:Np,1:Np)*x(1:Np);
2   for k=2:K
```

```
3       range=(k-1)*Np+1:k*Np
4          x(range)=M1(range,1:Np)*( x(range)...
5                           +LIFT*(Scale(range).*x(vmapP(range)) );
6   end
```

Here `LIFT, Scale`, and `vmapP` are a matrix operator, a scaling-factor array, and an index-of-neighboring-nodes array respectively. Essentially, instead of invoking the sparse-matrix multiplication routines, this algorithm rebuilds the entries of the matrix, while collects only the necessary data (using `vmapP`). This results in faster, dense-matrix multiplications. This implementation, "blockGS," only includes advective terms, since the inclusion of diffusive terms requires significant re-coding due to the intermediate q variable, and in practice would be implemented along with the function performing the matrix-free, $\mathbf{A}x$ multiplication. This Block GS preconditioner is expected to perform best for advection-dominated flows (Persson and Peraire, 2008), improving the rate of convergence more than the pure Jacobi preconditioner. The decrease in run-time for the new implementation is substantial. From $O(10^{-3})$-$O(10^{-4})$ to $O(10^{-5})$ seconds per multiplication, giving a 10-100 fold decrease in computational time. Hence, this type of implementation should be used for realistic applications.

With the Block GS implementation, it was also possible to include the flux contributions of the un-updated x vector. However, leaving the flux contributions of the un-updated x vector resulted in poorer performance, hence they were removed by a switch (incorporated through the scale-factor array).

Finally, in order to also include the diffusive effects with the block GS preconditioner, a final implementation where the entire lower-block matrix is pre-inverted was utilized. This was the standard implementation used for these studies, and all numbers reported use this implementation.

5.3.5 p-MG Preconditioner

The first p-MG implementation, "MG," was similar to the third implementations of both the Jacobi and GS preconditioners. A function computing $\mathbf{M}^{-1}x$ was passed to the solver. Here $\mathbf{M}^{-1} = \mathcal{R}_{1->p^h}\mathbf{M}_{p=1}^{-1}\mathcal{R}_{p^h->1}$, where $\mathcal{R}_{p^h->1}x$ **restricts** x from a higher order basis function ($p = p^h$) to a first order basis function ($p = 1$), and $\mathcal{R}_{1->p^h}x$ prolongate the solution. In this implementation, $\mathbf{M}_{p=1}^{-1}$ was pre-computed, so the only cost of the $\mathbf{M}^{-1}x$ calculations were due to the restriction/prolongation and matrix-vector multiplications.

Note, the projection/restriction operators are normally defined for a *modal* basis function, where $u(\vec{x}) = \sum_i u_i^M \psi_i(\vec{x})$. The restriction operator is then easily defined by simply truncating the number of modes. This is different from an interpolation, and caution is required when implementing a p-MG scheme when using a nodal basis.

The nodal basis used for this work can be presented as $\mathcal{V}u^M = u^N$, where $\mathcal{V}_{ij} = \psi_j(x_i)$ is a generalized Vandermonde matrix, and u^N is the approximate value of u at the nodal points x_i. Interpolation of the solution works as follows:

$$\begin{aligned} u^M &= \mathcal{V}_N^{-1}u^N \\ u_1^N &= \mathcal{V}_{N1}\hat{u}^M = \mathcal{V}_{N1}\mathcal{V}_N^{-1}u^N \\ u_1^M &= \mathcal{V}_1^{-1}u_1^N \\ u^N &= \mathcal{V}_{1N}u_1^M = \mathcal{V}_{1N}\mathcal{V}_1^{-1}u_1^N \end{aligned} \tag{5.6}$$

where $(\mathcal{V}_N)_{hj} = \psi_j(x_h^N)$, $(\mathcal{V}_{N1})_{ij} = \psi_j(x_i^1)$, $(\mathcal{V}_1)_{ik} = \psi_k(x_i^1)$, $(\mathcal{V}_{1N})_{jk} = \psi_k(x_j^N)$, $h = 1, ..., N$, $j = 1, ..., N$, $i = 1, 2, 3$, and $k = 1, 2, 3$. Here x^1 represents the nodal points for the $p = 1$ basis, and x^N represents the nodal points for the $p = N$ basis, and the subscript $(.)_1$ on u indicates the approximate solution on the $p = 1$ basis.

Interpolation is NOT the same as restriction/prolongation, however, because the interpolant fails to remove the modes that cannot be solved for on the coarser $p = 1$ grid. That is, in the interpolating case, the higher order basis is not contained in the lower order basis. This has consequences for the Galerkin formulation used where

the residuals on both the low and high order bases are set orthogonal to their own basis. In the interpolating case when solving on the coarse grid, one attempts to set information from the higher order basis orthogonal to the lower order basis. However the information contained in the higher order bases is not contained in the lower order basis, and the correction calculated on the coarser grid then attempts to correct for the errors from the higher order modes even though it cannot. If a collocation scheme was used instead, this interpolating strategy may be appropriate, but here we need to be more careful. Instead, the operations should be as follows:

$$\begin{aligned} u^M &= \mathcal{V}_N^{-1} u^N \\ u_{\mathcal{E}}^M &= \mathcal{E} u_N^M \\ u_{1,\mathcal{E}} &= \mathcal{V}_1 u_{\mathcal{E}}^M = \mathcal{V}_1 \mathcal{E} \mathcal{V}_N^{-1} u^N \\ u_1^M &= \mathcal{V}_1^{-1} u_1^N \\ e_{1,P}^M &= \mathcal{E}^T \hat{e}_1 = E^T V_1^{-1} e_1 \\ e_N^P &= V_N \hat{e}_1^P = V_N E^T V_1^{-1} e_1 \end{aligned} \tag{5.7}$$

The only difference from the case of the interpolation is that here we have included a $\mathcal{E}$ matrix. $\mathcal{E} \in \Re^{N,3}$ is basically a cutoff filter, and for this particular implementation,

$$\mathcal{E}_{i,j} = \begin{cases} 1 & (i,j) = [(1,1) \;\; (2,2) \;\; (N+2,3)] \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

With the $\mathcal{E}$ matrix included, the correct restriction/prolongation operator is defined.

A more realistic implementation was also attempted, where the residual $r = b - \mathbf{A}\tilde{x}$ instead is solved on the coarse grid $\tilde{e} = (\mathcal{R}^T \mathbf{A}_1^{-1} \mathcal{R}) r$, and the correction is applied as $x = e + \tilde{x}$. Some additional coding was required, but the function from the "MG" implementation could be re-used. The new MG implementation was combined with ILU(0) preconditioned GMRES(m). It was desirable to use BiCGSTAB(l) however the function from Sleijpen (2009) would not accept an initial guess, and hence could not be restarted in a loop. Hence, GMRES(m) was used instead. The algorithm was

as follows:

```
for i=1,2,...
   [X] = GMRES(A,B,RESTART,TOL,MAXIT,M1,M2,X); %Smoothing
   Resid=B-A*X;                                %Residual
   Resid_1=R*Resid;                            %Restriction
   E_1=inv(A_1)*Resid_1;                       %Coarse grid solution
   E=R'*E_1;                                   %Prolongation
   X=X+E;                                      %Correction
```

In one step, the scheme is $x = (\mathbf{I} - \mathcal{R}^T \mathbf{A}_1^{-1} \mathcal{R}\mathbf{A})x + \mathcal{R}^T \mathbf{A}_1^{-1} \mathcal{R} b$. Here $\mathcal{R}$ is a restriction operator and its transpose ,$\mathcal{R}^T$, is the prolongation operator; $\mathbf{A}_1$ is the matrix to solve the problem using $p = 1$ order basis functions, and $\mathbf{A}$ is the matrix to solve the problem for a higher order basis function; x is the approximate solution at the current iteration; and b is the right-hand side vector of the $\mathbf{A}x = b$ system.

5.3.6 Numerical Experiments

A brief description of the functions and scripts written for this study can be found in Appendix B.

A number of numerical benchmarks and tests were run, and each is briefly described below. The primary benchmark starts with a wide range of solver, preconditioner, and flow configurations to identify promising directions to investigate. The benchmarks that follow test progressively fewer combinations, until only two solver choices with one preconditioner remain. The performance of these two solver/preconditioner combinations are then analyzed to identify whether additional improvements can be made to the convergence rate. Finally, to see if there is any improvement is possible, a combined ILU(0) and MG preconditioner is tested.

The numerical experiments were conducted on a 2.4 GHz Intel desktop computer running Windows XP. The domain used has 757 triangles and 1162 faces, and uses

4^{th} order basis functions for a total of 11,355 degrees of freedom. The HDG implementation reduces the number of global unknowns to 5,810.

The primary benchmark was run with both and HDG and LDG discertizations. The magnitude of the velocity varies throughout the domain, and the scaling factor is taken as one for cases where advection is turned on, and zero when advection is turned off. The value taken for the diffusivity is taken as one when diffusion is on and zero when it is off.

All the LDG simulations are initialized with the previous timestep, whereas the HDG simulations are initialized using a zero vector for the input. At large values of timestep size, the initial guess vector will not have a large impact on the solution time. In the reported results the system is solved once for the LDG discretization, and three times for the HDG discretization. For the rest of the numerical tests, only the performance of LDG discretizations were examined.

Primary benchmark

The primary benchmark tests the performance of all the solvers for all the different preconditioners for different flow parameters, a total of 900 combinations. In summary, the following parameters were varied:

- Discretization: [LDG, HDG]
- Solvers: [MATLAB's "Slash"' solver, GMRES(m) with $m = 20$, BiCGSTAB(l) with $l = 10$, QMR]
- Preconditioners: All the preconditioners listed in §5.3.2.
- Advection: [No advection, advection]
- Diffusion: [No diffusion, Diffusion]
- dt = [0.2, 2, 20, 200, 2000]

Note that cases with large timestep sizes are examined because they are important to advance solutions toward steady-state, and this also follows the work of Persson

and Peraire (2008). This benchmark was run twice and the number of iterations were consistent.

GMRES(m) and BiCGSTAB(l) restart benchmark

The purpose of the "restart" benchmark was to find the best value of m and l for GMRES(m) and BiCGSTAB(l) respectively, with a total of 585 tests. This benchmark varied the following parameters:

- Discretization: LDG
- Solvers: [GMRES(m) with various m, BiCGSTAB(l) with various l]
- m, $2 \times l = [2, 4, 8, 10, 12, 14, 16, 18, 20, 24, 30, 40, 50]$
- Preconditioner: ILU(0).
- Advection: [No advection, advection]
- Diffusion: [No diffusion, Diffusion]
- dt = [0.2, 2, 20, 200, 2000]

This benchmark was run twice and the number of iterations were consistent.

BiCGSTAB(l) with $l = 5, 9$ ILU benchmark

Here the behavior BiCGSTAB(l)with $l = 5, 9$ using different ILU preconditioners was examined for advection-diffusion with a CFL number of 10,000. In this case, the computation time of the preconditioner was important, and hence was included in the benchmark time. The following parameters were varied:

- ILU factorization options

  setup.type = ['nofill', 'ilutp']

  setup.droptol=[10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}, 10^{-10}]

Convergence history test

In the convergence history test, GMRES(m) with $m = 20$, BiCGSTAB(l) with $l = 10$, and QMR are compared with and without the same ILU(0) or p-MG preconditioner for an LDG discretization. Only a timestep size of 200 is considered, and all three flow regimes are considered. The convergence history is plotted with and without the preconditioner (for both ILU(0) and p-MG), and the 50 largest and smallest scaled eigenvalues of the ILU(0) preconditioned and un-preconditioned $\mathbf{A}$ matrix are plotted on the complex plane.

MG with ILU(0) smoother

In this test, a proper MG scheme is combined with a preconditioned GMRES(m) smoother ,and the convergence rates are examined for all flow regimes with timestep size of 2000. For reference, all simulations are plotted along with both the convergence rate of ILU(0) preconditioned GMRES(m) and naive p-MG preconditioned GMRES(m). The following were varied:

- GMRES(m)preconditioner: [Naive MG, none, ILU(0)]
- Fine grid basis function order: [4, 2]

5.3.7 Discussion and Results

Primary benchmark results and discussion

The results of the primary benchmark for HDG can be found in Tables A.2 – A.4, and the results for LDG can be found in Tables A.5 – A.7. Note that the benchmark time results for MATLAB's "slash" operator are also included. Also note that the "slash" solver is not preconditioned, but the times recorded serve to quantify the variability of the benchmark time, as well as give a basis of comparison for a fast solver. MATLAB's "slash" operator is likely using a spare-direct solver for this problem. Note that BiCGSTAB(l) and GMRES(m) are competitive with the "slash" operator at small timestep sizes (or low CFL numbers). Also note, in the

"Summary" column, the iterations for QMR and BiCGSTAB(l) are multiplied by a factor of 2 because they require 2 matrix-vector (MV) multiplications per iteration, whereas GMRES(m) only requires one. Hence, the "Min MV" column plots the minimum matrix-vector multiplications, but only serves to give an approximate cost of the method, because both GMRES(m) and BiCGSTAB(l) become more expensive for larger m and l respectively.

A number of simulations did not converge within the specified maximum iteration count. This will be addressed once good solver-preconditioner combinations are found. In particular, the QMR solver was the least robust, failing to converge for the largest number of cases, whereas BiCGSTAB(l) appeared to be the most robust.

The traditional preconditioners do not generally perform as well as their block counterparts. While the traditional preconditioners tend to decrease the number of iterations till convergence over the unpreconditioned case, the block-preconditioners decrease the number of iterations by a larger factor.

As expected, the two Block Jacobi implementations and the block ILU preconditioner converge in a consistent number of iterations with the same residual, however the times for each vary by as much as a factor of 10 (see dt=2000, BiCGSTAB(l), Table A.4). Because the Jacobi2 implementation is up to 10 times slower than the Jacobi implementation for BiCGSTAB(l), the times for the GS preconditioner can be taken to be on the order of ten times smaller for BiCGSTAB(l) because the implementations are similar.

For some cases the Jacobi2 implementation converged in fewer iterations (for example see dt=2000, BiCGSTAB(l), Table A.4), but these are exceptional cases where the Jacobi2 implementation converged in one outer iteration of BiCGSTAB(l) before the other implementations, and this may be explained by rounding errors in the calculation of the residual, since the residual is used as a stopping criterion at the end of each outer iteration. The value of the residual is larger for the Jacobi2 implementation which converged in fewer iterations, and this supports the proposed hypothesis. However, the reason why the Jacobi2 implementation converged one outer iteration sooner is not completely clear.

Focusing on the number of iteration, where fewer iterations determine better performance, the Block GS preconditioner generally performs better than the Block Jacobi preconditioners for all flow regimes with both the LDG and HDG implementations. However, the GS preconditioner is outperformed by the ILU(0) preconditioner in all cases for the HDG discretization. For the LDG implementation, however, the ILU(0) preconditioner sometimes performs better, usually for smaller time-step sizes and more advective flows, while the GS preconditioner seems better for larger timesteps. Although, in the LDG implementation, the naive implementation of the MG preconditioner gives the best consistent performance. Thus, for the HDG implementation, the ILU(0) preconditioner seems to perform best, while for the LDG implementation the naive implementation of the MG preconditioner performs best.

The excellent performance of the naive implementation of the MG preconditioner for the LDG implementation was unexpected. Initially when the interpolating restriction/prolongation operators were used, poor performance was observed for the MG preconditioner. However, while the MG preconditioner improved the rate of convergence for the HDG implementation over the case where no preconditioner was used, it was consistently outperformed by the ILU(0) preconditioner. The mismatch between the performance of the MG preconditioner for the two different implementations indicate either that the good performance for the LDG implementation is special, or that the restriction/prolongation operators are incorrectly specified for the HDG implementation. Since the HDG method maps information from the interior of an element to its edges, the restriction/prolongation of the solution on the edges may not follow the same rules as the method discussed in Section 5.3.5 for the restriction/prolongation of the solution on the elements. The excellent performance for the MG preconditioner on the LDG discretization may be explained by the orthogonality of the Koornwinder modal basis and the linearity of the problem, in which case the MG preconditioner applied naively ($\mathbf{M}1\mathbf{A} = \mathbf{M}1b$) essentially solves the lower modes directly. For non-linear problems, the same result cannot be expected. Then, the linearity property may be lost for the HDG implementation, which could explain why the naive MG implementation does not perform equally well. In either case, further

investigation is warranted, but is beyond the scope of this thesis.

Comparing the performance of the HDG implementation to the LDG implementation, it can be seen the the HDG implementation converged for more cases than the LDG implementation. Also, keeping in mind that the HDG implementation solved the system three times whereas the LDG implementation solves the system only once, the HDG implementation tended to converge in fewer iterations and completed the calculations faster than the LDG implementation for the same preconditioner. However, due to the excellent performance of the MG preocnditioner, for some cases with timestep sizes larger than 20 (CFL larger than 100) the LDG implementation was found to be more efficient than the HDG implementation. This means that either a more competitive preconditioner for the HDG implementation needs to be found, or that, for an iterative solution method, there is no clear winner between the LDG and HDG method.

For small values of the timestep size, the ILU(0) preconditioned GMRES(m) or BiCGSTAB(l) solvers converge acceptably fast for both discretizations, and only problems with large timestep size still require a better preconditioner.

This section identified the ILU(0) and p–MG preconditioners using the GMRES(m) or BiCGSTAB(l) solvers as promising directions to investigate. Examining the results, it can also be seen that pure diffusive cases are more difficult to solve, and a better preconditioning scheme is needed to handle these cases. Additionally, the HDG discretization converged more robustly and faster than the LDG discretization when using the ILU(0) preconditioner, however it did not see a drastic improvement in performance using the MG preconditioner whereas the LDG discretization did.

GMRES(m) and BiCGSTAB(l) restart benchmark results and discussion

No clear winner for the choice of solver was evident from the primary benchmark, but both GMRES(m) and BiCGSTAB(l) performed better than QMR. Since these algorithms depend on m and l respectively, m and l were varied to see if an additional gain in performance could be realized while using the ILU(0) preconditioner. The results are reported in Tables A.8 – A.10.

For pure advection at small timestep sizes, GMRES(m) was slightly faster than BiCGSTAB(l) but the performance was comparable (note the residual is smaller for BiCGSTAB(l) for these cases). At large timestep sizes, BiCGSTAB(l) with $l \approx 7$ performed consistently well, converging within the specified iteration tolerance for all timestep sizes, whereas GMRES(m) failed to converge within the specified iteration tolerance for the largest timestep size ($dt = 2000$). Interestingly, BiCGSTAB(l) with large values of l seems less robust, since the cases with large l did not converge for all pure advective cases.

For pure diffusion, both GMRES(m) and BiCGSTAB(l) did not converge at large timestep sizes. The residuals were similar for both solvers at the the small timestep sizes although BiCGSTAB(l) had one order of magnitude smaller residuals at the largest timestep size, but the performance was comparable. At small timestep sizes, GMRES(m) was slightly faster, but the performance was again similar. Overall, the performance was poor for both solvers, and a better preconditioner is required for diffusive regimes when using a large timestep size.

The results for the advection-diffusion case were similar to the pure diffusion case, suggesting the flow regime chosen is more diffusion-dominated. Although, for the advection-diffusion case, fewer converged solutions resulted, specifically BiCGSTAB(l) failing to converge for more cases than the pure diffusive case. The final residual is smaller for GMRES(m) with $m = 50$ compared to BiCGSTAB(l) for all cases where the timestep size was smaller than 2000. However, at a timestep size of 2000, the smallest residual for BiCGSTAB(l) was an order of magnitude smaller than the smallest GMRES(m) residual. Regardless, a better preconditioner is required to handle the solution of the diffusive terms.

Overall, for GMRES(m) small values of m resulted in faster convergences, whereas larger values of GMRES(m) seemed to converge more robustly. For BiCGSTAB(l) $l \approx 7$ is a good choice. The total performance of the two solvers were similar, but because BiCGSTAB(l) was more robust, it is the preferred solver.

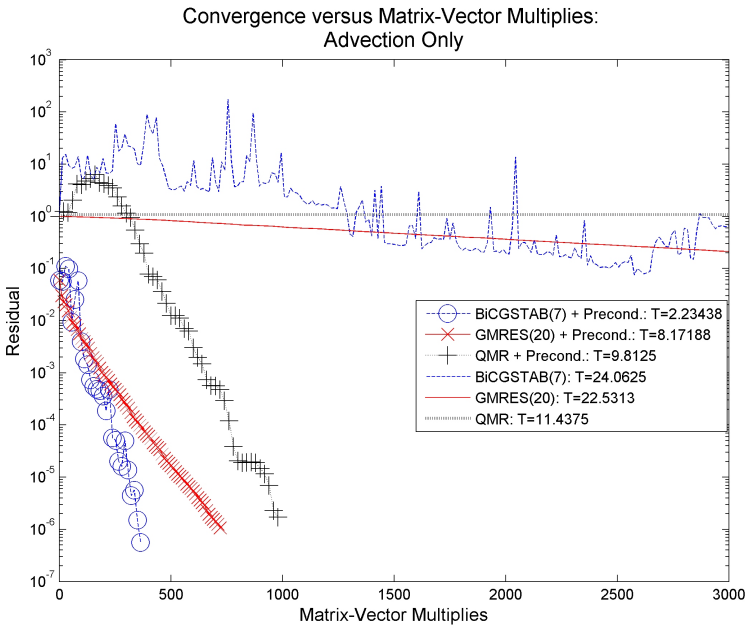

Figure 5-3: Residual history of different solvers with and without ILU(0) preconditioner for advection only flow regime with timestep size of 200

BiCGSTAB(l) with $l = 5, 9$ ILU benchmark

The ILU(0) preconditioner was superior in computational time compared to all the other ILU factorizations attempted.

Convergence history test results and discussion

The convergence histories using the ILU(0) preconditioner are plotted in Figures 5-3 to 5-5 for each flow regime, and the convergence histories using the p–MG preconditioner are plotted in Figures 5-6 to 5-8 for each flow regime.

The most notable feature is the rapid convergence of p–MG preconditioned GMRES(m) and BiCGSTAB(l). QMR does not see an improvement from the p–MG preconditioner, and this may be explained by the fact that the QMR implementation also needs $(\mathbf{M}1^T)^{-1}$ supplied to it. This suggests that either the implementation is incorrect, or that a naive implementation is not sufficient, and more care needs to be taken with the restriction/prolongation portion of this preconditioner when implementing the function performing the $(\mathbf{M}1^T)^{-1}x$ matrix-vector multiply. QMR con-

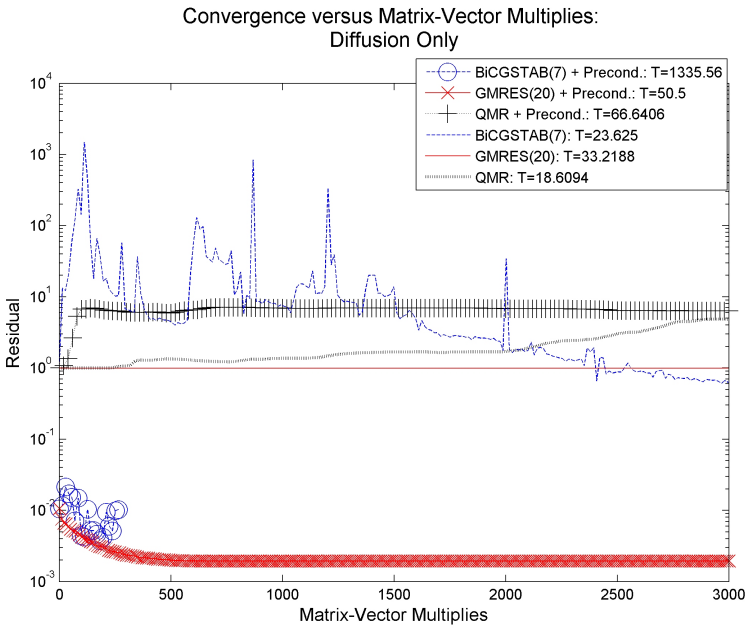

Figure 5-4: Residual history of different solvers with and without ILU(0) preconditioner for diffusion only flow regime with timestep size of 200

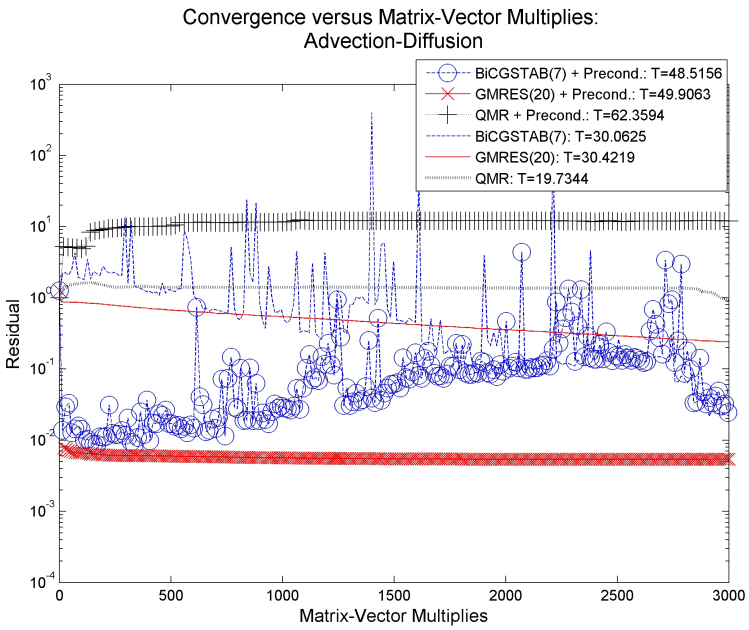

Figure 5-5: Residual history of different solvers with and without ILU(0) preconditioner for advection-diffusion flow regime with timestep size of 200

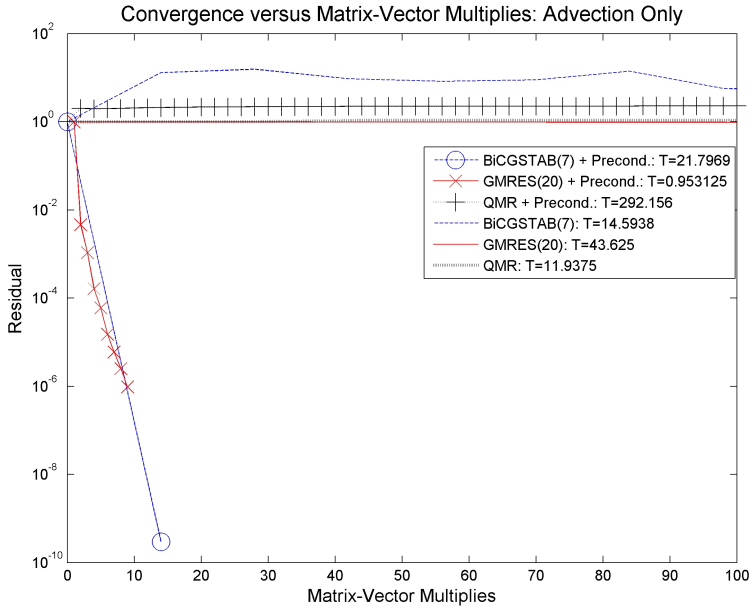

Figure 5-6: Residual history of different solvers with and without p–MG preconditioner for advection only flow regime with timestep size of 200

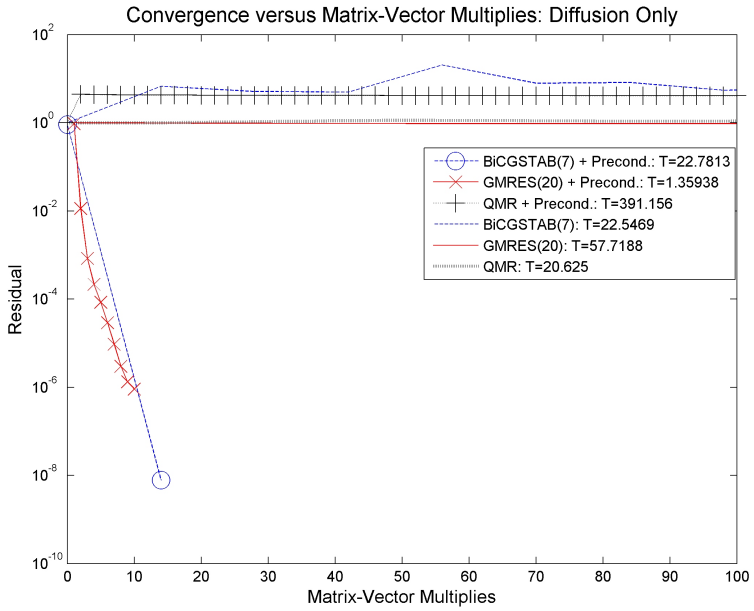

Figure 5-7: Residual history of different solvers with and without p–MG preconditioner for diffusion only flow regime with timestep size of 200

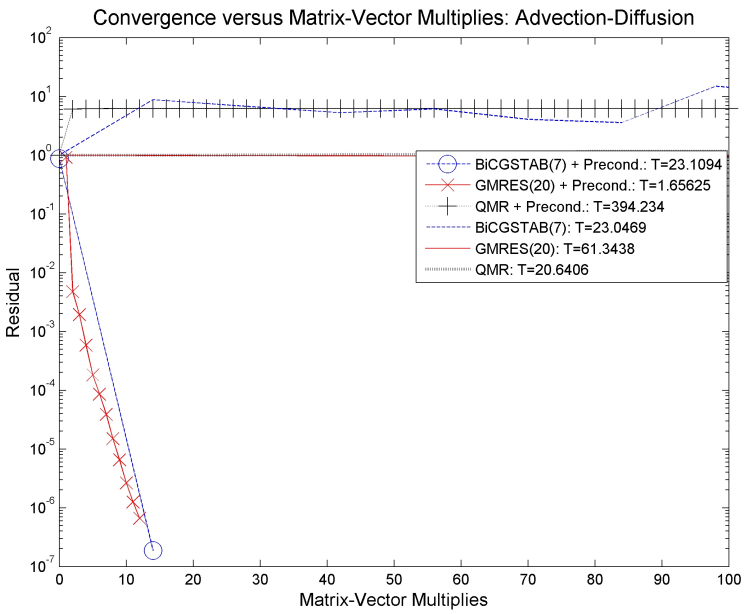

Figure 5-8: Residual history of different solvers with and without p–MG preconditioner for advection-diffusion flow regime with timestep size of 200

verged slowly and smoothly for most cases but diverged for some cases. GMRES(m) tended to converge smoothly and monotonically as expected, whereas BiCGSTAB(l) would converge erratically but with a general downward trend for most cases.

The p–MG preconditioner improved the convergence rate considerably for all flow regimes, with BiCGSTAB(l) converging within one outer iteration and GMRES(m) converging within 15 iterations. While these results are favorable, it is worth still considering the ILU(0) preconditioner, since the p-MG result may only hold for the special case of linear equations, and a more general preconditioner capable of handling non-linear problems is desired. While it is expected that the p-MG preconditioner would improve the rate of convergence for diffusive problems, the improvement for the advective case was not expected, and may be due to the linearity of the problem. Also, the p-MG preconditioner did not improve the convergence of the HDG discretization, hence we examine the performance of the ILU(0) preconditioner next.

The ILU(0) preconditioner only improved the convergence rate for the pure-advective case for all the solvers, although the improvement was not as impressive as the p–MG preconditioner. The ILU(0) preconditioner seemed to improve the initial

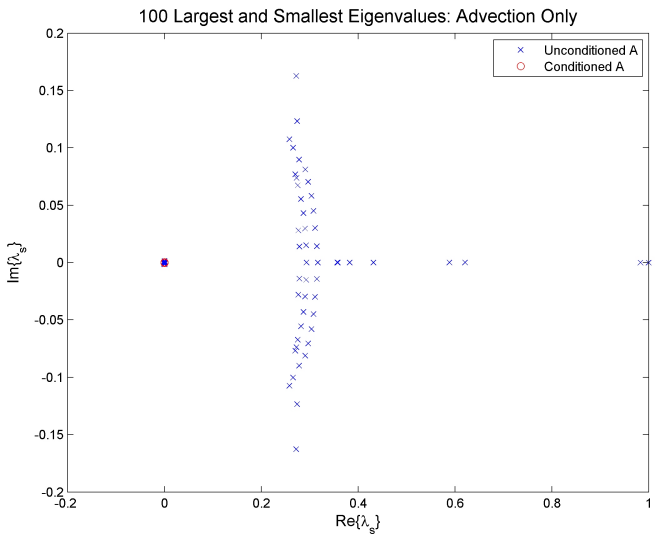

Figure 5-9: Eigenvalues of conditioned and unconditioned A matrices for pure advection with timestep size 200. The eigenvalues λ_s are normalized by $\lambda_s = \lambda/\Lambda_{max}$, where $\Lambda_{max} = \max \Re\{\lambda\} - \min \Re\{\lambda\}$ is the maximum range of the real component of the eigenvalues of both matrices

reduction of the residual for all cases, but for the advection-diffusion case, the rate of the convergence for GMRES(m) is clearly slower for the preconditioned matrix. To gain insight into why this happens, we examine the eigenvalues of the preconditioned and unpreconditioned matrices, which are reported in Figures 5-9 to 5-11.

Note that the eigenvalues in 5-9 to 5-11 are normalized by the maximum range of the real component of the eigenvalues of both matrices. From Trefethen and Bau (1997), the convergence of GMRES(m) is improved when the eigenvalues are localized and do not surround the origin. From Figure 5-9, it can be seen that the ILU(0) preconditioner seems to localize the eigenvalues of the original system considerably, whereas Figures 5-10 and 5-11 still show that the eigenvalues have large imaginary components. Whether or not the preconditioned eigenvalues surround the origin are not clear from Figures 5-9 to 5-11, and thus a different normalization of the eigenvalues are plotted in Figures 5-12 to 5-14. Here the eigenvalues are normalized by the maximum absolute value of the real part of the eigenvalue belonging to the specific matrix.

From Figure 5-12 we see that for pure advection the normalized preconditioned

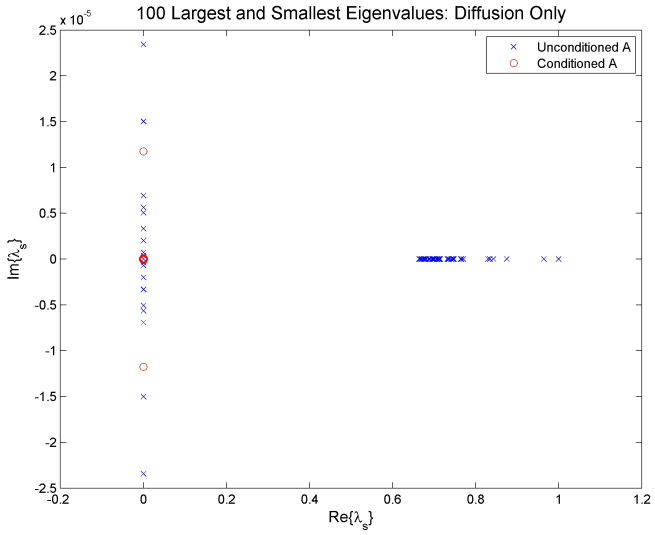

Figure 5-10: Eigenvalues of conditioned and unconditioned A matrices for pure diffusion with timestep size 200. The eigenvalues λ_s are normalized by $\lambda_s = \lambda/\Lambda_{max}$, where $\Lambda_{max} = \max \Re\{\lambda\} - \min \Re\{\lambda\}$ is the maximum range of the real component of the eigenvalues of both matrices

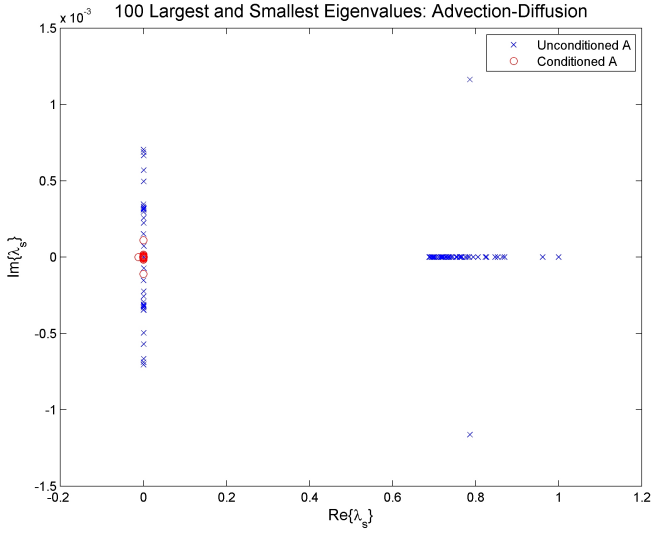

Figure 5-11: Eigenvalues of conditioned and unconditioned A matrices for advection-diffusion with timestep size 200. The eigenvalues λ_s are normalized by $\lambda_s = \lambda/\Lambda_{max}$, where $\Lambda_{max} = \max \Re\{\lambda\} - \min \Re\{\lambda\}$ is the maximum range of the real component of the eigenvalues of both matrices

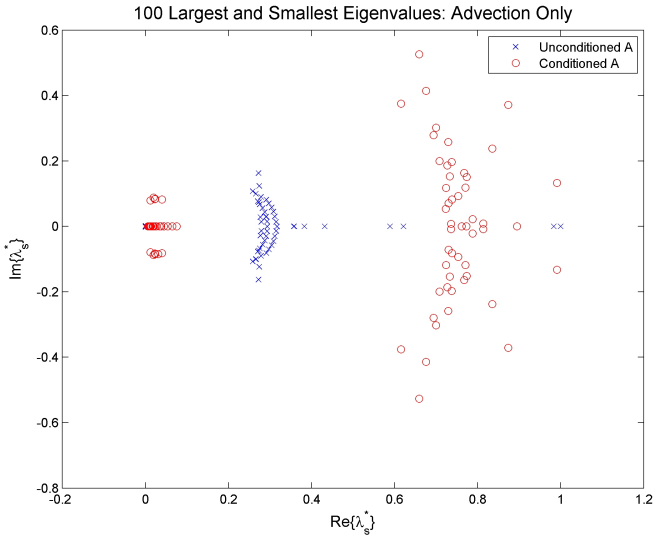

Figure 5-12: Eigenvalues of conditioned and unconditioned A matrices for pure advection with timestep size 200. The eigenvalues λ_s^* are normalized by $\lambda_s^* = \lambda/\lambda_{max}$, where $\lambda_{max} = \max \Re\{\lambda\}$ is the maximum absolute value of the real part of the eigenvalue belonging to the specific matrix

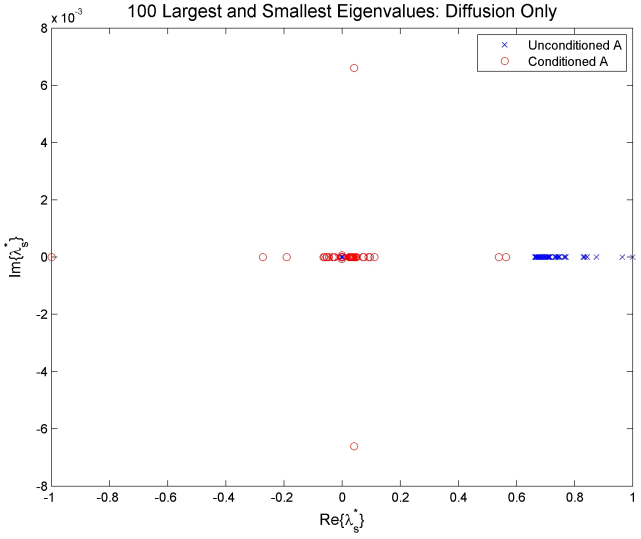

Figure 5-13: Eigenvalues of conditioned and unconditioned A matrices for pure diffusion with timestep size 200. The eigenvalues λ_s^* are normalized by $\lambda_s^* = \lambda/\lambda_{max}$, where $\lambda_{max} = \max \Re\{\lambda\}$ is the maximum absolute value of the real part of the eigenvalue belonging to the specific matrix

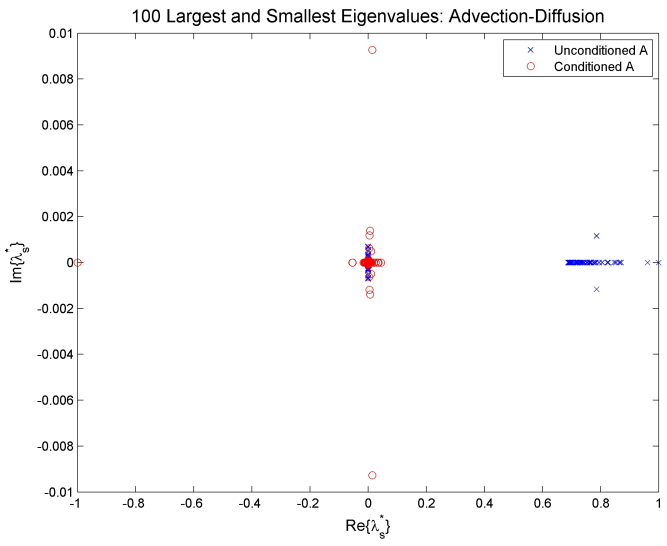

Figure 5-14: Eigenvalues of conditioned and unconditioned A matrices for advection-diffusion with timestep size 200. The eigenvalues λ_s^* are normalized by $\lambda_s^* = \lambda/\lambda_{max}$, where $\lambda_{max} = \max \Re\{\lambda\}$ is the maximum absolute value of the real part of the eigenvalue belonging to the specific matrix

eigenvalues do not surround the origin and are generally more localized even though the relative imaginary components of the eigenvalues are larger. In cases with diffusion, Figure 5-13 and 5-14 show that the preconditioned eigenvalues surround the origin in both cases, and that the relative size of the imaginary components of the preconditioned matrix are larger. More importantly, it also pushes the smallest eigenvalues closer to zero such that the ratio $\lambda_{max}/\lambda_{min}$ is increased. Therefore the ILU(0) preconditioner does not favorably scale the matrix for the cases where diffusion is involved. This suggests that a better preconditioner is required to handle the diffusive part of the flow.

MG with preconditioned GMRES(m) smoother results and discussion

All results are reported in Appendix C. Note that in the legend "MG" refers to the properly implemented MG preconditioner with a GMRES(m) smoother, where the preconditioner used for the smoother is indicated in the caption, and "MG naive" refers to the naive MG implementation used during the Primary benchmark.

In all cases, the naive MG preconditioner is considerably better than any other

combination. It is rivaled only by the the proper MG implementation when the GMRES(m) smoother is preconditioned by the naive MG preconditioner, in which case the naive MG preconditioner is doing most of the work. Considering that the primary benchmark indicated that the naive MG preconditioner did not work for the HDG implementation, the cases where the naive MG preconditioner is not used are also examined.

Considering the case where the fine-grid disrectization uses fourth order basis functions, the convergence is examined. For pure advection, if the GMRES(m) smoother is not preconditioned, the solver diverges! However, the ILU(0) preconditioned GMRES(m) smoother combined with the proper MG scheme increases the rate of convergence over using ILU(0) preconditioned GMRES(m) without the MG correction. The pure diffusion case divergences with the proper MG preconditioner using either unpreconditioned or ILU(0) preoconditioned GMRES(m). The advection-diffusion case diverges when GMRES(m) is preconditioned with ILU(0)$\dot{\text{W}}$hen GMRES(m) is not preconditioned and only uses the proper MG scheme, it converges faster than if GMRES(m) uses only the ILU(0) preconditioner (that is without the MG correction). These disheartening results indicate that a single solution scheme does not suffice, and a good preconditioner for the pure diffusive case has not been found. In all cases, the residual is increased during the MG correction, even though the overall rate of convergence is increased. The increase of the residual after the MG correction is suspected to be due to the large jump in grid sizes between the $p = 4$ and $p = 1$ disrectizations. Hence, the MG scheme is also examined for a fine-grid discretization using $p = 2$ order basis functions.

Considering the case where the fine-grid discretization uses second order basis functions, and still ignoring cases where the naive MG preconditioner is used, the convergence is examined. In this case, the ILU(0) preconditioned GMRES(m) smoother combined with the proper MG preconditioner gives better results than using only ILU(0) preconditioned GMRES(m). The pure advection case converges within 600 matrix-vector multiplies, and the advection-diffusion case converges within 100. For the pure diffusion case, the simulation does not converge within 1000 matrix-vector

multiplies, but the final residual is nearly two orders of magnitude lower than when using only the ILU(0) preconditioned GMRES(m) solver, and the rate of convergence is faster. The advection-diffusion case did not see an increase in the residual after the MG correction, whereas the pure advection and pure diffusion cases still saw the increase. This result shows that when the difference between the order of the basis used on the coarse and fine grid discretizations are not as large, improved convergence can be realized using the proper MG implementation. This suggests that a hierarchal p–MG scheme (V or W MG for example) could be the best choice.

5.4 Conclusions and Recommendations

The best preconditioner for the LDG discretization is the naive p-MG preconditioner and for the HDG implementation the best preconditioner found was the ILU(0) preconditioner, although proper p–MG schemes were not examined for HDG. For both discretizations, the BiCGSTAB(l) solver seemed to be the most robust, and gave the best consistent performance.

It was found that the HDG discretized matrices were faster to solve with fewer iterations in cases where the naive p–MG preconditioner was not used for the LDG discretization. With the naive p–MG preconditioner, the LDG discretized matrices can be solved more efficiency for timestep sizes larger than 20 (CFL approximately 100) than the HDG discretized matrices.

Values of $l \approx 7$ seemed to work best for BiCGSTAB(l) whereas for GMRES(m) smaller values of m resulted in faster convergence, and larger values of m resulted in more robust performance.

For properly implemented MG schemes, this works suggests a hierarchal p–MG scheme, where the change in p between levels is not too large, may improve the rate of convergence when using an ILU(0) preconditioned GMRES(m) smoother.

It is recommended that a naive MG preconditioner is examined in detail in order to, extend it for use with the HDG discretization and general non-linear problems. The benchmarks for LDG should also be repeated for an HDG discretization in order

to find a good preconditioner for HDG. Further examination of the proper p-MG implementation for HDG is also warranted if the naive p–MG implementation is found not to work. A hierarchal proper p–MG scheme might be necessary.

The work presented here enables the efficient implicit solution of advection-diffusion problems for solving biogeochemical reactions in the ocean. Also, equations such as the Incompressible Navier Stokes equations can now be solved efficiently with the procedure described.

Chapter 6

Conclusions

The purpose of this thesis is to identify promising numerical methods that are suitable to multiscale ocean predictions. In order to fulfill this purpose, current efforts towards creating new ocean models are reviewed, an understanding of the most promising methods used by other researchers is developed, the most promising existing methods are studied and applied to idealized cases, new methods are incubated and evaluated by solving biogeochemical advection-diffusion-reactions equations, and efficient solver/preconditioner combinations for inverting DG FEM matrices are identified.

From our quantitative incubation of numerical schemes, a number of recommendations on the tools necessary to solve dynamical equations for multiscale ocean predictions are provided, and a summary follows.

6.1 Summary of Results

Most of the second generation ocean models reviewed use some form of the FEM. The FEM models are more sophisticated than their FV counterparts, but due to the added complexity are less mature. The sophistication of the FEM models arise from the freedom for higher order schemes, and the freedom to choose the space of the solution and test functions. The FV method models reviewed are more mature and ready to be used for realistic problems.

The DG FEM is a promising numerical method for developing the next generation ocean models if the efficiency constraints can be overcome. The DG FEM method offers efficient data structures for parallel implementations, higher order accuracy, geometric flexibility enabling sophisticated adaptive algorithms, and superconvergence properties for dispersion and dissipation, making the method particularly well suited to advection-dominated flows.

The DG FEM method was implemented for ocean biogeochemical reaction equations, which to our knowledge, is the first time this has been done. The numerical implementation was verified using a number of test cases, and the LSRK time integration scheme was found to be more accurate than the first order Euler time integration scheme.

A purely advective test case (the advection of a cosine bell) was used to demonstrate that a higher order scheme can be more accurate, more efficient, and use fewer degrees of freedom than a lower order scheme. It was concluded that high and low order schemes should also be compared on a efficiency-accuracy basis to complement the DOF-efficiency based comparison.

It shown that such a p-adaptive scheme using different orders of basis functions for different constituents on the same element is promising for improving the efficiency and the accuracy of the solution. However, it was also shown that such schemes need to consider the cost of additional volume and edge interpolation operations when formulating the adaptation criterion.

It was argued that adaptive algorithms are necessary to resolve important small scale features which would go unnoticed if a coarse non-adaptive scheme was used.

While explicit time integration schemes were sufficient for the advective operators, the stability constraints associated with the diffusive terms were prohibitively expensive. It was concluded that implicit time integration schemes were necessary when small values of grid Peclet number (large values of κ) is required.

For an LDG discretization, it was found that a naive p– MG preconditioner was optimum, whereas the ILU(0) preconditioner was the best preconditioner found for an HDG discretization. For both discretizations, the BiCGSTAB(l) solver with $l \approx 7$

offered consistently efficient and robust performance, being slightly better than a GMRES(m) solver, and much better than a QMR solver. HDG discretized systems were found to be faster to solve with fewer iterations than LDG discretized systems whenever the naive LDG p–MG preconditioner was not used. The naively p-MG preconditioned LDG solves were sometimes faster than the HDG solves for timestep sizes larger than approximately 20 (or CFL number approximately 100). It was argued that a hierarchal p–MG scheme may improve the rate of convergence when using an ILU(0) preconditioned GMRES(m) smoother.

6.2 Recommendations

It is recommended that a mature FV model such as SUNTANS, FVCOM, or a mature FEM models such as SELFE, ADCIRC, or FEOM is used if unstructured grids are necessary for immediate application. Applications that require unstructured grids would normally be found in regions with complex geometries or bottom topography.

Adopting a sophisticated adaptive model such as SLIM or ICOM for near future use is recommended. The flexibility and accuracy of these adaptive models promise to widen the range of ocean processes that can be studied, specifically processes that depend on multiple scales.

Additional examination of DG FEMs for ocean simulations is recommended due to the advantages of these methods. In particular, HDG methods seem to be a promising avenue to explore. While the efficiency of DG methods may not be as good as compact FD schemes, the additional geometric flexibility of DG methods warrant additional attention. It is recommended that DG should be studied on adaptive structured grids for ocean models and their efficiency compared to traditional structured grid schemes. This approach also enables the accurate treatment of complex geometries by having an unstructured grid at boundaries for accuracy, while maintaining the an efficient structured mesh for the bulk of the unknown in the interior.

It is recommended that high-order quadrature-free adaptive algorithms are used whenever possible for optimum accuracy and efficiency. Also, explicit time integration

is recommended for advective operators, whereas implicit time integration schemes are recommended for diffusive operators due to prohibitively expensive numerical stability constraints for small grid Peclet numbers.

A naive MG preconditioner is examined in detail in order to extend it for use with the HDG discretization and general non-linear problems. Further examination of the proper p-MG implementation for HDG and non-linear problems is also warranted if the naive p–MG implementation performs poorly. A hierarchal proper p–MG scheme may be necessary.

6.3 Future work

A next step is to be able to solve both the physics and biology using HDG, so as to explore biogeochamical ocean processes. This could allow two-dimensional idealized studies of coupled physics-biology, possibly aiding parameter selection for realistic three-dimensional simulations. Additionally, this code would serve as a test-bed for adaptive algorithms.

The HDG method will also be explored. Using HDG discretized projection methods may yield an efficient solution method for solving the Incompressible Navier Stokes equations.

The solution of two-dimensional physics could be extended to three dimensions. Adaptive oct-tree algorithms can be examined for their efficiency and compared to more standard unstructured adaptive algorithms.

Appendix A

Tables

Table A.1: Detailed table of Second generation ocean models

	Equations				Time Integration				Vertical Grid				Method				Adaptive Finite Elements			Does not apply to FV		Ravard Thomas Hydrostatic			Turbulence Closure	Advection Scheme	Wetting/Drying	Free Surface	Biological Models	Ice Model	Parallel	Available for download	
	PE	SW	Im Ex	Semi	Theta	Subcy	Sig	Z	Sd	IP	U	FE	FV	Spect	-	P1DGP1	P1-P1	P1NC-P1	NHS	HS	QHS	-											
ICOM	x			x			x			x	x			h	x	x	x	x	x		-	T-L	P-G	-	-	-				x	-	NY
SLIM	?	x	?						x	x	x			h	x	x	x	x	?		?	M-Y	Various	x	x	NPDZ				x	x	YES
FEOM	x			x			x	x		x	x						x	x			?	?	Various*	?	x	x	?			x	MPI	YES
Delft/Finlab	?			?			?				x	x		?					?	?	?	RANS?	Eul-Lag	?	x	x	?			?	?	?
UnTRIM*		3D		x			x					x+FD							x	x	x	Filter?	FL-TVD	?	x	x					MPI	YES
SEOM	T	x	?			x				x	x			h-p					x	x	x	?	?	?	x	x					MPI	BUY
ADCIRC		x	x			?	2d??					DG								x	x	?	Eul-Lag	x	x					?	upgrading from CG to DG
ELCIRC		x		x				x				x								x	x	?	Eul-Lag	x	x					?	YES
SELFE		3D		x			x	x				x						similar		x	x	?	GeLS	Eul-Lag	x	x	ECO-SELFE				MPI	YES
SUNTANS	x		x		x		x	x				x							x			LES	Perot	x	x					MPI	YES
MIT-CGM	x		x						x										x		x	**	Vtest		x					x	YES	-multiple schemes
FVCOM	x			R-K?			x			?		x	x	x					new	x		GOTM	?	?	x	x					MPI	YES?
RiCOM	?						?							?							?											?

Sd-shaved T-test

sig-sigma T-L: Temmekes and Lumley (1972)
Temmekes, H., and J. L. Lumley, 1972: A First Course in Turbulence.
MIT Press, 300 pp.
*Orthogonal unstructured grid, solve various non-linear D-E's on unstructured grid
**: Red/GM - various 'packages' available

M-Y-Mellor-Yamada 2.5 Turbulence closure scheme
GeLS-Generic Length Scale
GOTM-General Ocean Turbulence Model (Burchard, 2002)

UNTRIM Unstructured Tidal Residual Intertidal Mudflat model

Various*: T-G, CN-GLS, FCT (Lohner)
Conserv: Seems to be Eulerian-Lagrangian scheme, but unsure_Confirmed with SELFE paper
FL-TVD: Flux-limiter Total Variation Diminishing
Perot - advection scheme of Perot (2000)
Vtest-Various schemes being currently tested
GLS- Galerkin least squares
T-G-Taylor Galerkin
CN-Crank Nicolson
FCT-Flux Corrected transport (by Lohner)
P-G: Petrov Galerkin
Eul-Lag: Eulerian-Lagrangian

NY-not yet

Table A.2: Primary Preconditioner/Solver benchmark results for $\kappa = 1, V_{scale} = 0$ using HDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

Slash	GmresR				Qmr				BiCgstab				Summary	
Time	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Min Time*	*Min MV*
dt=0.2		dt=0.2				dt=0.2				dt=0.2			dt=0.2	
0.4063	None	0.5156	182	8.7E-07	None	0.4063	118	8.2E-07	None	0.3281	90	4.0E-09	0.3281	180
0.4375	Upper	0.2188	105	8.8E-07	Upper	0.4844	58	3.3E-07	Upper	0.2344	30	3.0E-08	0.2188	60
0.4375	Lower	0.2031	105	8.4E-07	Lower	0.5000	58	3.0E-07	Lower	0.2656	30	2.9E-08	0.2031	60
0.4375	Jacobi	0.2656	129	8.9E-07	Jacobi	0.3281	75	8.8E-07	Jacobi	0.2656	60	4.1E-10	0.2656	120
0.4219	ILU0	0.1406	87	8.6E-07	ILU0	0.3594	30	4.6E-07	ILU0	0.3125	30	9.0E-12	0.1406	60
0.4375	BlockILU	0.3125	119	6.0E-07	BlockILU	0.4375	66	6.4E-07	BlockILU	0.2969	50	8.0E-07	0.2969	100
0.4375	BlockJacobi	0.3750	119	6.0E-07	BlockJacobi	0.6563	66	6.4E-07	BlockJacobi	0.5156	50	8.0E-07	0.3750	100
0.4375	BlockJacobi2	0.6563	119	6.0E-07	BlockJacobi2	1.0781	66	6.4E-07	BlockJacobi2	3.0781	50	8.0E-07	0.6563	100
0.4375	GS	0.2969	100	9.7E-07	GS	0.5	48	7.1E-07	GS	3.71875	30	2.4E-08	0.2969	60
0.4063	Multigrid	2.7500	89	1.7E-07	Multigrid	27.7969	128	2.3E-03	Multigrid	87.1563	30	1.8E-12	2.7500	60
													0.1406	**60**
dt=2		dt=2				dt=2				dt=2			dt=2	
0.4219	None	0.8750	229	8.4E-07	None	0.5938	160	9.8E-07	None	0.4688	100	5.5E-07	0.4688	200
0.4375	Upper	0.3906	135	6.6E-07	Upper	0.7656	87	9.3E-07	Upper	0.5000	60	7.6E-10	0.3906	120
0.4063	Lower	0.4063	135	7.5E-07	Lower	0.7344	89	9.9E-07	Lower	0.4531	60	7.3E-10	0.4063	120
0.4063	Jacobi	0.4688	177	7.8E-07	Jacobi	0.5938	125	7.5E-07	Jacobi	0.3281	60	8.3E-07	0.3281	120
0.4375	ILU0	0.2813	109	9.4E-07	ILU0	0.5625	51	6.8E-07	ILU0	0.3438	30	7.5E-08	0.2813	60
0.4219	BlockILU	0.4688	162	8.5E-07	BlockILU	0.7031	110	7.9E-07	BlockILU	0.4219	60	1.6E-07	0.4219	120
0.4063	BlockJacobi	0.7344	162	8.5E-07	BlockJacobi	0.9844	110	7.8E-07	BlockJacobi	0.5938	60	1.6E-07	0.5938	120
0.4063	BlockJacobi2	1.1406	162	8.5E-07	BlockJacobi2	1.8438	110	7.8E-07	BlockJacobi2	3.6406	60	1.6E-07	1.1406	120
0.4375	GS	0.5781	132	8.9E-07	GS	0.85938	84	7.8E-07	GS	7.09375	60	3.4E-10	0.5781	120
0.4219	Multigrid	6.8906	136	9.3E-07	Multigrid	28.1094	45	7.2E-03	Multigrid	166.0156	60	9.6E-10	6.8906	120
													0.2813	**60**
dt=20		dt=20				dt=20				dt=20			dt=20	
0.3906	None	1.3594	403	9.9E-07	None	0.6563	180	1.5E-04	None	0.7656	180	5.8E-07	0.7656	360
0.4063	Upper	0.8281	213	7.8E-07	Upper	1.3438	177	1.3E-06	Upper	0.5938	90	1.2E-07	0.5938	180
0.4063	Lower	0.8906	213	8.5E-07	Lower	1.3906	179	2.1E-06	Lower	0.5625	90	2.2E-07	0.5625	180
0.4375	Jacobi	1.0625	322	9.4E-07	Jacobi	0.7031	180	3.4E-05	Jacobi	0.7969	150	8.3E-08	0.7969	300
0.4063	ILU0	0.4844	147	8.0E-07	ILU0	0.9531	87	6.3E-07	ILU0	0.5156	60	4.7E-09	0.4844	120
0.4063	BlockILU	1.0938	257	8.9E-07	BlockILU	1.1875	180	2.3E-06	BlockILU	0.8125	120	3.7E-08	0.8125	240
0.4063	BlockJacobi	1.3750	257	8.9E-07	BlockJacobi	1.6719	180	2.3E-06	BlockJacobi	1.1250	120	3.7E-08	1.1250	240
0.4063	BlockJacobi2	2.0938	257	8.9E-07	BlockJacobi2	2.9063	180	2.3E-06	BlockJacobi2	7.0938	120	3.8E-08	2.0938	240
0.3906	GS	1.0625	194	7.5E-07	GS	1.4E+00	146	9.5E-07	GS	10.3125	90	6.6E-09	1.0625	180
0.4063	Multigrid	20.3281	294	9.0E-07	Multigrid	28.0156	86	4.4E-02	Multigrid	349.4688	130	9.6E-07	20.3281	260
													0.4844	**120**
dt=200		dt=200				dt=200				dt=200			dt=200	
0.3906	None	7.0000	1775	9.7E-07	None	0.6250	180	4.6E-02	None	2.0469	520	6.1E-07	2.0469	1040
0.4063	Upper	3.2969	663	9.6E-07	Upper	1.3750	173	9.9E-02	Upper	1.3438	240	2.1E-07	1.3438	480
0.4063	Lower	3.2031	664	9.9E-07	Lower	1.4063	163	8.7E-02	Lower	1.4063	240	2.0E-07	1.4063	480
0.4375	Jacobi	4.9375	1228	9.2E-07	Jacobi	0.7656	180	3.1E-02	Jacobi	1.6563	410	4.7E-07	1.6563	820
0.4063	ILU0	1.5938	331	9.8E-07	ILU0	1.8906	180	5.5E-06	ILU0	0.9063	120	6.9E-07	0.9063	240
0.4063	BlockILU	4.2656	859	9.8E-07	BlockILU	1.2188	180	1.3E-02	BlockILU	1.5625	310	4.7E-07	1.5625	620
0.3906	BlockJacobi	5.3125	859	9.8E-07	BlockJacobi	1.6563	180	1.3E-02	BlockJacobi	2.3750	310	4.7E-07	2.3750	620
0.4375	BlockJacobi2	8.2500	859	9.8E-07	BlockJacobi2	2.9063	180	1.3E-02	BlockJacobi2	17.7656	310	5.0E-07	8.2500	620
0.4219	GS	3.2188	550	9.8E-07	GS	1.7E+00	169	4.7E-03	GS	23.6563	210	7.5E-08	3.2188	420
0.3906	Multigrid	102.3438	1248	9.7E-07	Multigrid	28.0469	180	2.5E-01	Multigrid	1087.8594	410	7.5E-07	######	820
													0.9063	**240**
dt=2000		dt=2000				dt=2000				dt=2000			dt=2000	
0.3906	None	14.0781	3660	2.7E-02	None	0.6250	180	3.7E-01	None	4.0938	1040	8.3E-07	4.0938	2080
0.4063	Upper	16.5625	3104	1.0E-06	Upper	1.3594	166	8.9E-01	Upper	2.6875	470	5.2E-07	2.6875	940
0.4063	Lower	18.5156	3520	9.9E-07	Lower	1.4219	75	8.8E-01	Lower	2.4844	430	2.0E-07	2.4844	860
0.4063	Jacobi	14.8594	3660	1.0E-05	Jacobi	0.7344	180	3.5E-01	Jacobi	4.3438	870	6.6E-07	4.3438	1740
0.4063	ILU0	7.1875	1275	9.9E-07	ILU0	1.9531	180	4.6E-03	ILU0	1.4844	230	1.4E-07	1.4844	460
0.4063	BlockILU	18.5000	3630	1.0E-06	BlockILU	1.1094	180	3.0E-01	BlockILU	3.3438	580	3.9E-07	3.3438	1160
0.4063	BlockJacobi	24.5156	3630	1.0E-06	BlockJacobi	1.6250	180	3.0E-01	BlockJacobi	5.2813	580	3.9E-07	5.2813	1160
0.3750	BlockJacobi2	36.9063	3631	9.9E-07	BlockJacobi2	2.9219	180	3.0E-01	BlockJacobi2	34.2500	600	3.1E-07	34.2500	1200
0.3906	GS	17.0625	2696	9.2E-07	GS	1.8E+00	98	1.0E+00	GS	44.7969	400	3.2E-07	17.0625	800
0.4063	Multigrid	314.6406	3660	2.1E-05	Multigrid	27.9688	126	7.8E-01	Multigrid	3439.9063	1320	5.4E-05	######	2640
													1.4844	**460**

Table A.3: Primary Preconditioner/Solver benchmark results for $\kappa = 0, V_{scale} = 1$ using HDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

Slash	GmresR				Qmr				BiCgstab				Summary	
Time	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Min Time*	*Min MV*
dt=0.2		dt=0.2				dt=0.2				dt=0.2			dt=0.2	
0.4375	None	0.5781	212	9.9E-07	None	0.6250	158	4.3E-07	None	0.5938	90	1.0E-07	0.5781	180
0.4219	Upper	0.2031	96	9.6E-07	Upper	0.3750	38	9.7E-07	Upper	0.2500	30	5.1E-11	0.2031	60
0.4375	Lower	0.1250	90	3.4E-07	Lower	0.2813	31	5.1E-07	Lower	0.2188	30	2.1E-13	0.1250	60
0.4375	Jacobi	0.1719	105	8.1E-07	Jacobi	0.2500	49	4.8E-07	Jacobi	0.1875	30	5.5E-09	0.1719	60
0.4219	ILU0	0.0938	78	1.7E-07	ILU0	0.2188	18	1.1E-07	ILU0	0.2500	30	6.0E-16	0.0938	36
0.4375	BlockILU	0.1406	98	7.5E-07	BlockILU	0.2813	40	8.5E-07	BlockILU	0.1719	30	2.5E-10	0.1406	60
0.4375	BlockJacobi	0.2188	98	7.5E-07	BlockJacobi	0.4063	41	3.8E-07	BlockJacobi	0.2969	30	2.5E-10	0.2188	60
0.4063	BlockJacobi2	0.4063	98	7.5E-07	BlockJacobi2	0.7344	41	3.8E-07	BlockJacobi2	1.8906	30	6.6E-10	0.4063	60
0.4219	GS	0.2656	86	2.0E-07	GS	0.32813	27	4.2E-07	GS	3.734375	30	6.3E-15	0.2656	54
0.4219	Multigrid	2.9844	94	9.8E-07	Multigrid	27.8750	17	1.1E-01	Multigrid	85.8281	30	6.6E-11	2.9844	60
													0.0938	36
dt=2		dt=2				dt=2				dt=2			dt=2	
0.4219	None	2.0938	644	1.0E-06	None	0.6406	165	8.2E-03	None	1.3750	280	2.7E-07	1.3750	560
0.4219	Upper	0.6875	197	8.8E-07	Upper	1.1250	134	4.9E-07	Upper	0.5938	80	6.8E-07	0.5938	160
0.4219	Lower	0.6094	170	7.2E-07	Lower	0.9063	111	6.4E-07	Lower	0.4844	70	5.6E-07	0.4844	140
0.4375	Jacobi	0.7813	254	9.0E-07	Jacobi	0.7500	176	2.9E-06	Jacobi	0.5469	110	5.9E-07	0.5469	220
0.4375	ILU0	0.2500	99	3.2E-07	ILU0	0.4531	41	3.1E-07	ILU0	0.3125	30	2.4E-12	0.2500	60
0.4375	BlockILU	0.9844	180	7.4E-07	BlockILU	1.3281	125	8.8E-07	BlockILU	0.8281	80	6.0E-08	0.8281	160
0.4375	BlockJacobi	0.8125	180	7.4E-07	BlockJacobi	1.2031	125	9.1E-07	BlockJacobi	0.8281	80	6.0E-08	0.8125	160
0.4688	BlockJacobi2	1.2656	180	7.4E-07	BlockJacobi2	2.0781	125	9.1E-07	BlockJacobi2	4.8438	80	6.0E-08	1.2656	160
0.4219	GS	0.4688	124	6.8E-07	GS	0.70313	67	8.6E-07	GS	4.859375	40	6.2E-07	0.4688	80
0.4219	Multigrid	8.0000	153	8.7E-07	Multigrid	27.8906	21	8.4E-02	Multigrid	165.6719	60	1.5E-08	8.0000	120
													0.2500	60
dt=20		dt=20				dt=20				dt=20			dt=20	
0.4219	None	14.3438	3660	5.9E-06	None	0.5938	158	4.0E-01	None	7.2813	1800	5.8E-06	7.2813	3600
0.4375	Upper	6.5469	1356	9.8E-07	Upper	1.3281	163	3.9E-02	Upper	3.4219	560	6.5E-07	3.4219	1120
0.4219	Lower	5.6563	1160	9.8E-07	Lower	1.4844	174	5.0E-02	Lower	2.6406	460	4.9E-07	2.6406	920
0.4375	Jacobi	8.1875	2065	9.9E-07	Jacobi	0.7188	165	2.0E-01	Jacobi	3.9531	820	2.4E-07	3.9531	1640
0.4219	ILU0	1.4531	305	9.5E-07	ILU0	1.9219	179	5.4E-06	ILU0	0.9375	130	4.9E-08	0.9375	260
0.4375	BlockILU	11.1094	1522	9.7E-07	BlockILU	2.0469	179	5.8E-02	BlockILU	5.9688	580	7.0E-07	5.9688	1160
0.4375	BlockJacobi	9.2813	1522	9.7E-07	BlockJacobi	1.6719	177	7.5E-02	BlockJacobi	5.2344	580	7.0E-07	5.2344	1160
0.4688	BlockJacobi2	15.0938	1522	9.7E-07	BlockJacobi2	2.8906	177	7.5E-02	BlockJacobi2	32.7188	570	6.5E-07	15.0938	1140
0.4375	GS	4.0625	691	9.9E-07	GS	1.6E+00	178	6.0E-03	GS	32.7344	290	7.3E-07	4.0625	580
0.4219	Multigrid	50.3125	644	9.6E-07	Multigrid	27.9844	55	7.7E-01	Multigrid	772.3438	290	1.7E-07	50.3125	580
													0.9375	260
dt=200		dt=200				dt=200				dt=200			dt=200	
0.4219	None	13.0938	3660	1.5E-02	None	0.5781	171	8.2E-01	None	8.0781	1800	1.5E-02	8.0781	3600
0.4219	Upper	20.0000	3060	8.4E-01	Upper	1.4063	17	9.6E-01	Upper	10.9531	1800	3.8E+01	10.9531	3600
0.4375	Lower	18.8750	3660	7.1E-01	Lower	1.3594	110	8.9E-01	Lower	10.5469	1800	2.5E+00	10.5469	3600
0.4688	Jacobi	14.0313	3640	4.7E-01	Jacobi	0.7031	151	8.9E-01	Jacobi	8.5156	1800	2.5E+01	8.5156	3600
0.4219	ILU0	21.6094	3620	8.3E-05	ILU0	1.9063	179	1.9E-01	ILU0	10.1406	1570	1.5E-06	10.1406	3140
0.4375	BlockILU	28.3594	2260	5.4E-01	BlockILU	2.0313	3	9.6E-01	BlockILU	19.3594	1800	5.0E+00	19.3594	3600
0.4375	BlockJacobi	24.0625	2260	5.4E-01	BlockJacobi	1.6563	6	9.5E-01	BlockJacobi	16.8750	1800	5.0E+00	16.8750	3600
0.4375	BlockJacobi2	36.2031	2260	5.4E-01	BlockJacobi2	2.8750	6	9.5E-01	BlockJacobi2	102.7969	1800	4.8E+00	######	3600
0.4375	GS	22.8750	3640	7.4E-01	GS	1.8E+00	127	9.5E-01	GS	238.4219	1800		######	3600
0.4219	Multigrid	308.5781	3660	1.6E-04	Multigrid	28.0781	37	9.2E-01	Multigrid	4739.0469	1800	5.4E-06	######	3600
													8.0781	3140
dt=2000		dt=2000				dt=2000				dt=2000			dt=2000	
0.4219	None	15.1250	3660	3.8E-01	None	0.5938	173	8.8E-01	None	52.4844	1800		52.4844	3600
0.4375	Upper	19.7344	3560	9.8E-01	Upper	1.3281	3	9.9E-01	Upper	10.2344	1800	1.3E+02	10.2344	3600
0.4375	Lower	19.6719	3420	1.4E+00	Lower	1.4063	121	9.7E-01	Lower	9.8281	1800	2.2E+01	9.8281	3600
0.4375	Jacobi	15.7188	3660	5.6E-01	Jacobi	0.7188	89	9.4E-01	Jacobi	7.3750	1800	4.9E+03	7.3750	3600
0.4375	ILU0	21.5000	2220	1.8E+00	ILU0	1.8750	38	9.8E-01	ILU0	11.6719	1800	4.6E-02	11.6719	3600
0.4375	BlockILU	27.7813	2560	5.7E-01	BlockILU	2.0469	3	9.6E-01	BlockILU	117.9375	1800		######	3600
0.4375	BlockJacobi	24.2500	2560	5.7E-01	BlockJacobi	1.6563	5	9.6E-01	BlockJacobi	106.6719	1800		######	3600
0.4375	BlockJacobi2	37.6719	2220	5.7E-01	BlockJacobi2	2.8750	5	9.6E-01	BlockJacobi2	190.0469	1800		######	3600
0.4219	GS	22.9531	2760	1.2E+00	GS	1.6E+00	82	9.7E-01	GS	284.5625	1800	3.9E+03	######	3600
0.4375	Multigrid	310.3438	3160	1.4E-01	Multigrid	28.0469	20	9.3E-01	Multigrid	4672.2500	1800	3.2E+00	######	3600
													7.3750	3600

Table A.4: Primary Preconditioner/Solver benchmark results for $\kappa = 1, V_{scale} = 1$ using HDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

Slash	GmresR				Qmr				BiCgstab				Summary	
Time	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Min Time*	*Min MV*
dt=0.2		dt=0.2				dt=0.2				dt=0.2			dt=0.2	
0.4219	None	0.6563	188	9.0E-07	None	0.5000	142	9.2E-07	None	0.4531	90	5.0E-09	0.4531	180
0.4063	Upper	0.2656	107	8.2E-07	Upper	0.5156	58	6.1E-07	Upper	0.2813	30	4.6E-08	0.2656	60
0.4375	Lower	0.2656	105	7.4E-07	Lower	0.4844	55	6.4E-07	Lower	0.2500	30	2.4E-08	0.2500	60
0.4375	Jacobi	0.3438	131	9.7E-07	Jacobi	0.3750	86	6.6E-07	Jacobi	0.3750	60	9.3E-10	0.3438	120
0.4375	ILU0	0.1563	88	9.5E-07	ILU0	0.3750	31	8.1E-07	ILU0	0.3438	30	1.8E-12	0.1563	60
0.4375	BlockILU	0.2969	119	9.7E-07	BlockILU	0.5000	73	7.4E-07	BlockILU	0.2500	30	7.2E-07	0.2500	60
0.4375	BlockJacobi	0.4063	119	9.7E-07	BlockJacobi	0.7344	73	8.0E-07	BlockJacobi	0.3594	30	7.2E-07	0.3594	60
0.4375	BlockJacobi2	0.6563	119	9.7E-07	BlockJacobi2	1.2344	73	8.0E-07	BlockJacobi2	2.0000	30	7.2E-07	0.6563	60
0.4219	GS	0.3750	99	8.9E-07	GS	0.54688	49	4.6E-07	GS	3.75	30	3.2E-09	0.3750	60
0.4219	Multigrid	3.2656	95	5.4E-07	Multigrid	28.0156	26	2.4E-03	Multigrid	86.7969	30	3.1E-10	3.2656	60
													0.1563	60
dt=2		dt=2				dt=2				dt=2			dt=2	
0.4219	None	0.8281	269	8.3E-07	None	0.6563	177	8.8E-06	None	0.5625	120	1.2E-07	0.5625	240
0.4375	Upper	0.4531	142	6.9E-07	Upper	0.8438	99	4.9E-07	Upper	0.4531	60	2.3E-09	0.4531	120
0.4375	Lower	0.4219	132	7.5E-07	Lower	0.7344	87	6.8E-07	Lower	0.3906	60	2.9E-10	0.3906	120
0.4375	Jacobi	0.5938	197	9.7E-07	Jacobi	0.7031	160	8.0E-07	Jacobi	0.5000	90	9.0E-09	0.5000	180
0.4375	ILU0	0.2969	108	9.8E-07	ILU0	0.6563	56	5.0E-07	ILU0	0.3438	30	5.2E-07	0.2969	60
0.4375	BlockILU	0.6094	171	8.8E-07	BlockILU	0.9844	146	1.0E-06	BlockILU	0.4531	70	3.1E-07	0.4531	140
0.4375	BlockJacobi	0.7500	171	8.8E-07	BlockJacobi	1.3750	146	9.9E-07	BlockJacobi	0.6719	70	3.1E-07	0.6719	140
0.4375	BlockJacobi2	1.2188	171	8.8E-07	BlockJacobi2	2.3281	146	9.9E-07	BlockJacobi2	4.2656	70	3.1E-07	1.2188	140
0.4219	GS	0.5781	127	7.8E-07	GS	0.82813	84	9.6E-07	GS	7.078125	60	5.1E-11	0.5781	120
0.4219	Multigrid	8.4063	157	7.5E-07	Multigrid	28.0781	26	2.4E-02	Multigrid	166.0000	60	7.9E-08	8.4063	120
													0.2969	60
dt=20		dt=20				dt=20				dt=20			dt=20	
0.3906	None	3.0469	832	9.8E-07	None	0.6094	176	4.8E-02	None	1.8125	380	6.2E-07	1.8125	760
0.4375	Upper	1.9531	430	9.6E-07	Upper	1.3906	166	2.1E-03	Upper	1.2500	190	2.6E-07	1.2500	380
0.4375	Lower	1.4688	340	9.8E-07	Lower	1.4688	175	7.2E-04	Lower	0.9375	160	1.6E-07	0.9375	320
0.4063	Jacobi	2.8438	733	1.0E-06	Jacobi	0.7344	178	2.3E-02	Jacobi	1.6406	330	1.8E-07	1.6406	660
0.4375	ILU0	0.8281	207	8.3E-07	ILU0	1.8281	165	6.4E-07	ILU0	0.7031	90	2.7E-07	0.7031	180
0.4063	BlockILU	2.3281	510	9.4E-07	BlockILU	1.2188	178	8.7E-03	BlockILU	1.4531	220	3.8E-07	1.4531	440
0.4375	BlockJacobi	3.1875	510	9.4E-07	BlockJacobi	1.6563	178	8.7E-03	BlockJacobi	1.9375	220	3.8E-07	1.9375	440
0.4063	BlockJacobi2	4.6719	510	9.4E-07	BlockJacobi2	2.8750	178	8.7E-03	BlockJacobi2	13.0938	220	3.8E-07	4.6719	440
0.4219	GS	1.6719	291	1.0E-06	GS	1.6E+00	180	1.4E-04	GS	14.7344	130	2.9E-07	1.6719	260
0.4063	Multigrid	60.0625	757	9.9E-07	Multigrid	27.9844	53	1.7E-01	Multigrid	955.3438	360	4.2E-07	60.0625	720
													0.7031	180
dt=200		dt=200				dt=200				dt=200			dt=200	
0.3906	None	14.0313	3660	8.7E-04	None	0.5625	75	6.7E-01	None	5.0781	1140	5.4E-07	5.0781	2280
0.4063	Upper	18.2656	3454	9.9E-07	Upper	1.4063	116	4.9E-01	Upper	3.6250	590	8.1E-07	3.6250	1180
0.4219	Lower	12.4375	2316	1.0E-06	Lower	1.3438	66	3.7E-01	Lower	2.8906	480	1.4E-07	2.8906	960
0.4063	Jacobi	15.1406	3660	2.6E-04	Jacobi	0.7031	178	7.9E-01	Jacobi	4.8438	1050	9.2E-07	4.8438	2100
0.4063	ILU0	5.1406	896	1.0E-06	ILU0	2.0469	178	2.4E-01	ILU0	1.6875	250	2.5E-07	1.6875	500
0.4063	BlockILU	18.8906	3660	9.6E-06	BlockILU	1.1250	172	5.1E-01	BlockILU	4.2500	740	9.6E-07	4.2500	1480
0.3906	BlockJacobi	23.2188	3660	9.6E-06	BlockJacobi	1.6563	172	5.0E-01	BlockJacobi	6.8750	740	9.6E-07	6.8750	1480
0.4063	BlockJacobi2	37.4063	3660	9.6E-06	BlockJacobi2	2.9531	172	5.0E-01	BlockJacobi2	42.5938	740	9.9E-07	42.5938	1480
0.3906	GS	10.7969	1743	9.9E-07	GS	1.6E+00	169	1.8E-01	GS	43.8125	390	5.7E-07	10.7969	780
0.3906	Multigrid	308.9531	3660	3.0E-04	Multigrid	28.1875	19	9.2E-01	Multigrid	3377.0313	1280	4.5E-07	######	2560
													1.6875	500
dt=2000		dt=2000				dt=2000				dt=2000			dt=2000	
0.4063	None	13.9688	3660	2.5E-01	None	0.5781	73	7.5E-01	None	4.5469	1500	1.7E-07	4.5469	3000
0.4063	Upper	18.8438	3660	4.2E-02	Upper	1.4063	72	9.5E-01	Upper	4.1875	740	7.5E-07	4.1875	1480
0.4063	Lower	18.7969	3660	2.7E-02	Lower	1.3281	34	9.2E-01	Lower	3.4688	610	8.9E-07	3.4688	1220
0.4063	Jacobi	15.0781	3660	7.1E-02	Jacobi	0.7656	24	8.7E-01	Jacobi	6.7813	1420	9.4E-07	6.7813	2840
0.4219	ILU0	21.7969	3660	5.4E-04	ILU0	1.8750	0	1.0E+00	ILU0	2.2344	340	3.6E-09	2.2344	680
0.4063	BlockILU	18.1875	3660	5.2E-02	BlockILU	1.1563	75	8.3E-01	BlockILU	5.3906	960	2.3E-07	5.3906	1920
0.4375	BlockJacobi	24.1875	3660	5.2E-02	BlockJacobi	1.6406	142	8.2E-01	BlockJacobi	9.0156	960	2.3E-07	9.0156	1920
0.4063	BlockJacobi2	37.4531	3660	5.2E-02	BlockJacobi2	2.9063	142	8.2E-01	BlockJacobi2	54.4375	950	4.5E-07	54.4375	1900
0.3906	GS	22.9375	3660	1.0E-02	GS	1.7E+00	3	9.7E-01	GS	54.5469	490	1.6E-07	54.5469	980
0.3906	Multigrid	311.4688	3660	4.5E-02	Multigrid	28.0781	34	9.8E-01	Multigrid	4527.0156	1740	6.6E-07	######	3480
													2.2344	680

Table A.5: Primary Preconditioner/Solver benchmark results for $\kappa = 1, V_{scale} = 0$ using LDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

Slash	GmresR				Qmr				BiCgstab				Summary	
Time	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Min Time*	*Min MV*
dt=0.2		dt=0.2				dt=0.2				dt=0.2			dt=0.2	
0.5781	None	0.5156	62	1.0E-06	None	0.5781	43	9.3E-07	None	0.4375	30	4.0E-08	0.4375	60
0.5938	Upper	0.2813	40	7.9E-07	Upper	0.9688	29	7.2E-07	Upper	0.4844	20	5.2E-07	0.2813	40
0.5938	Lower	0.2969	36	9.3E-07	Lower	0.7344	24	4.8E-07	Lower	0.3594	20	4.8E-09	0.2969	36
0.6094	Jacobi	0.3594	49	9.6E-07	Jacobi	0.6406	40	4.1E-07	Jacobi	0.4063	30	2.0E-07	0.3594	49
0.5938	ILU0	0.1406	27	7.5E-07	ILU0	0.4219	9	5.2E-07	ILU0	0.3906	10	2.0E-11	0.1406	18
0.5938	BlockILU	0.6406	43	7.8E-07	BlockILU	1.5000	35	4.0E-07	BlockILU	1.0781	30	4.0E-09	0.6406	43
0.5938	BlockJacobi	0.6094	43	7.8E-07	BlockJacobi	1.6563	35	4.0E-07	BlockJacobi	1.1094	30	4.0E-09	0.6094	43
0.5938	BlockJacobi2	1.1094	43	7.8E-07	BlockJacobi2	2.4844	35	4.0E-07	BlockJacobi2	9.4844	30	3.4E-09	1.1094	43
0.5938	GS	0.8750	34	5.2E-07	GS	1.90625	19	7.1E-07	GS	20.0625	20	4.0E-10	0.8750	34
0.5938	Multigrid	1.1875	29	5.9E-07	Multigrid	15.1563	60	1.3E-01	Multigrid	29.6719	10	1.3E-11	1.1875	20
													0.1406	**18**
dt=2		dt=2				dt=2				dt=2			dt=2	
0.7031	None	2.1094	248	9.9E-07	None	0.9063	60	1.1E-02	None	1.3281	110	8.4E-07	1.3281	220
0.6406	Upper	2.0000	173	9.8E-07	Upper	1.9063	55	2.1E-02	Upper	2.2969	140	2.7E-07	2.0000	173
0.6563	Lower	1.2031	107	9.6E-07	Lower	1.5156	58	7.0E-03	Lower	1.2656	80	4.2E-07	1.2031	107
0.6406	Jacobi	2.1094	243	9.9E-07	Jacobi	0.8906	59	5.1E-02	Jacobi	1.9844	160	7.0E-07	1.9844	243
0.6563	ILU0	18.1719	1220	4.4E-06	ILU0	2.4688	55	2.2E-01	ILU0	6.8125	370	4.5E-07	6.8125	740
0.6563	BlockILU	3.7188	168	9.8E-07	BlockILU	2.8594	57	2.2E-02	BlockILU	4.7969	120	4.4E-07	3.7188	168
0.6406	BlockJacobi	3.4531	168	9.8E-07	BlockJacobi	2.6094	51	2.2E-02	BlockJacobi	4.2188	120	4.4E-07	3.4531	168
0.6563	BlockJacobi2	5.6563	168	9.8E-07	BlockJacobi2	4.1094	51	2.2E-02	BlockJacobi2	37.2344	120	5.3E-07	5.6563	168
0.6406	GS	3.6094	93	9.5E-07	GS	5.39063	60	7.5E-04	GS	67.203125	70	7.2E-07	3.6094	93
0.6719	Multigrid	1.4063	31	7.0E-07	Multigrid	15.1875	1	1.4E+00	Multigrid	30.0313	10	3.6E-11	1.4063	20
													1.2031	**20**
dt=20		dt=20				dt=20				dt=20			dt=20	
0.6563	None	10.5000	1220	5.4E-02	None	0.8125	38	7.4E-01	None	6.4531	600	3.4E-06	6.4531	1200
0.6250	Upper	15.6875	1220	5.5E-05	Upper	1.9219	12	7.3E-01	Upper	9.7656	600	3.4E-06	9.7656	1200
0.6406	Lower	14.7813	1220	5.2E-05	Lower	1.4531	9	1.9E+00	Lower	8.5625	600	1.2E-06	8.5625	1200
0.6719	Jacobi	10.8438	1220	9.0E-05	Jacobi	1.0469	34	1.0E+00	Jacobi	7.3125	600	9.2E-06	7.3125	1200
0.6250	ILU0	18.0469	1220	3.9E-03	ILU0	2.5313	16	6.0E+00	ILU0	10.9688	600	1.1E-02	10.9688	1200
0.6719	BlockILU	28.8750	1220	9.0E-06	BlockILU	2.9375	59	4.1E-01	BlockILU	23.7813	600	1.7E-06	23.7813	1200
0.6406	BlockJacobi	26.6094	1220	9.0E-06	BlockJacobi	2.6563	59	4.7E-01	BlockJacobi	20.7344	600	1.7E-06	20.7344	1200
0.6250	BlockJacobi2	44.6563	1220	8.8E-06	BlockJacobi2	4.1719	59	4.7E-01	BlockJacobi2	184.6875	600	3.2E-06	######	1200
0.6250	GS	56.1250	1220	4.8E-06	GS	5.4E+00	44	6.7E-01	GS	536.7656	560	7.4E-07	######	1120
0.6250	Multigrid	1.5469	32	5.5E-07	Multigrid	15.2031	0	1.5E+01	Multigrid	30.1875	10	5.4E-11	1.5469	20
													1.5469	**20**
dt=200		dt=200				dt=200				dt=200			dt=200	
0.7344	None	10.7031	1180	6.6E+00	None	0.7500	45	8.1E+00	None	6.8906	600	6.6E+00	6.8906	1200
0.6563	Upper	15.3438	1220	1.9E-05	Upper	1.8906	12	7.7E+00	Upper	9.9688	600	3.8E-05	9.9688	1200
0.6563	Lower	14.6406	1220	4.0E-05	Lower	1.5625	11	1.7E+01	Lower	8.5625	600	7.4E-05	8.5625	1200
0.7188	Jacobi	11.5781	1220	1.3E-05	Jacobi	0.9844	40	3.5E+00	Jacobi	6.8281	600	2.3E-05	6.8281	1200
0.6250	ILU0	18.1563	1220	2.7E-03	ILU0	2.4531	18	4.5E+01	ILU0	10.8438	600	1.0E-01	10.8438	1200
0.6719	BlockILU	30.3906	1220	4.0E-06	BlockILU	2.9219	34	3.9E+00	BlockILU	23.9844	600	6.3E-06	23.9844	1200
0.6406	BlockJacobi	25.9844	1220	4.0E-06	BlockJacobi	2.6719	34	3.8E+00	BlockJacobi	20.9063	600	6.3E-06	20.9063	1200
0.6250	BlockJacobi2	44.1250	1220	4.1E-06	BlockJacobi2	4.1563	34	3.8E+00	BlockJacobi2	184.6875	600	6.0E-06	######	1200
0.6563	GS	56.2031	1220	8.3E-06	GS	5.3E+00	60	5.3E+00	GS	576.7813	600	2.0E-04	######	1200
0.6563	Multigrid	1.5625	32	9.2E-07	Multigrid	15.2656	0	1.5E+02	Multigrid	30.3594	10	8.1E-09	1.5625	20
													1.5625	**20**
dt=2000		dt=2000				dt=2000				dt=2000			dt=2000	
0.7188	None	10.8906	1220	6.7E+01	None	0.8438	45	8.1E+01	None	6.5000	600	2.8E+01	6.5000	1200
0.6406	Upper	15.8125	1220	1.2E-05	Upper	1.8438	36	7.2E+01	Upper	9.7188	600	3.4E-06	9.7188	1200
0.6563	Lower	14.6406	1220	1.0E-05	Lower	1.5938	12	2.0E+02	Lower	8.3594	600	9.5E-05	8.3594	1200
0.6406	Jacobi	11.4531	1220	7.0E-06	Jacobi	0.9219	48	2.3E+01	Jacobi	7.2969	600	2.5E-06	7.2969	1200
0.6250	ILU0	17.7656	1220	2.6E-03	ILU0	2.5000	60	1.0E+03	ILU0	10.9531	600	2.4E-04	10.9531	1200
0.6563	BlockILU	29.6406	1220	1.4E-06	BlockILU	2.8750	41	4.3E+01	BlockILU	23.0781	580	9.6E-07	23.0781	1160
0.6563	BlockJacobi	25.9219	1220	1.4E-06	BlockJacobi	2.6406	41	4.5E+01	BlockJacobi	19.8906	580	9.6E-07	19.8906	1160
0.6250	BlockJacobi2	44.8281	1220	1.4E-06	BlockJacobi2	4.1719	41	4.5E+01	BlockJacobi2	154.0625	500	9.8E-07	######	1000
0.6250	GS	56.0000	1220	1.6E-06	GS	5.4E+00	60	6.5E+01	GS	575.5938	600	2.5E-06	######	1200
0.6250	Multigrid	1.6406	33	6.3E-07	Multigrid	15.2344	4	1.4E+03	Multigrid	30.4531	10	3.2E-11	1.6406	20
													1.6406	**20**

Table A.6: Primary Preconditioner/Solver benchmark results for $\kappa = 0, V_{scale} = 1$ using LDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

Slash	GmresR				Qmr				BiCgstab				Summary	
Time	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Min Time*	*Min MV*
dt=0.2		dt=0.2				dt=0.2				dt=0.2			dt=0.2	
0.2969	None	0.4063	53	7.9E-07	None	0.3750	39	9.9E-07	None	0.4531	20	2.1E-07	0.3750	40
0.2500	Upper	0.3594	39	9.2E-07	Upper	0.5156	25	7.8E-07	Upper	0.3594	20	7.0E-08	0.3594	39
0.2813	Lower	0.2656	34	8.3E-07	Lower	0.3906	16	7.7E-07	Lower	0.3125	20	9.3E-11	0.2656	32
0.2500	Jacobi	0.2813	45	7.9E-07	Jacobi	0.3281	30	6.7E-07	Jacobi	0.3906	20	3.8E-07	0.2813	40
0.2656	ILU0	0.1250	24	6.6E-07	ILU0	0.2031	5	2.6E-08	ILU0	0.2656	10	4.4E-14	0.1250	10
0.2813	BlockILU	0.2188	26	9.9E-07	BlockILU	0.2656	7	3.0E-07	BlockILU	0.4063	10	1.5E-13	0.2188	14
0.2500	BlockJacobi	0.1875	26	9.9E-07	BlockJacobi	0.3594	7	3.0E-07	BlockJacobi	0.5156	10	1.5E-13	0.1875	14
0.2344	BlockJacobi2	0.2813	26	9.9E-07	BlockJacobi2	0.3906	7	3.0E-07	BlockJacobi2	2.4531	10	1.8E-13	0.2813	14
0.2344	GS	0.2031	25	1.6E-07	GS	0.29688	6	1.4E-08	GS	3.4375	10	2.3E-15	0.2031	12
0.2031	Multigrid	0.4063	23	5.8E-07	Multigrid	11.5938	1	2.2E-02	Multigrid	28.5000	10	3.5E-12	0.4063	20
													0.1250	**10**
dt=2		dt=2				dt=2				dt=2			dt=2	
0.2500	None	2.0625	317	9.7E-07	None	0.4844	50	1.3E-01	None	1.4063	140	4.5E-07	1.4063	280
0.1875	Upper	10.1719	380	3.2E+02	Upper	0.9531	6	2.2E-01	Upper	105.2500	600		######	1200
0.2031	Lower	11.1094	1200	7.1E+01	Lower	1.1250	0	2.2E-01	Lower	7.0469	600	5.1E+00	7.0469	1200
0.2500	Jacobi	4.8281	703	9.9E-07	Jacobi	0.5938	58	1.0E-01	Jacobi	3.2500	290	7.0E-07	3.2500	580
0.2188	ILU0	0.2188	31	7.9E-07	ILU0	0.3281	14	4.4E-07	ILU0	0.2969	10	3.4E-08	0.2188	20
0.2031	BlockILU	0.6094	41	6.6E-07	BlockILU	1.1406	25	4.9E-07	BlockILU	0.8906	20	3.5E-12	0.6094	40
0.2188	BlockJacobi	0.6094	41	6.6E-07	BlockJacobi	0.9844	25	4.9E-07	BlockJacobi	0.7500	20	3.5E-12	0.6094	40
0.1875	BlockJacobi2	0.5938	41	6.6E-07	BlockJacobi2	0.9531	25	4.9E-07	BlockJacobi2	4.5938	20	3.5E-12	0.5938	40
0.1875	GS	0.3906	33	5.9E-07	GS	0.71875	17	1.4E-07	GS	3.609375	10	6.3E-09	0.3906	20
0.1875	Multigrid	0.5469	25	9.1E-07	Multigrid	11.2656	0	2.2E-01	Multigrid	29.2656	10	7.5E-14	0.5469	20
													0.2188	**20**
dt=20		dt=20				dt=20				dt=20			dt=20	
0.2500	None	7.9219	1220	1.9E-03	None	0.4688	5	1.8E+00	None	5.5625	600	1.4E-03	5.5625	1200
0.1875	Upper	10.6875	1220	8.3E+19	Upper	0.0625	0	2.2E+00	Upper	72.8438	600		72.8438	1200
0.2031	Lower	10.9375	1220	1.2E+17	Lower	1.0000	0	2.2E+00	Lower	84.7344	600		84.7344	1200
0.2500	Jacobi	8.6406	1220	4.6E-01	Jacobi	0.5469	0	2.2E+00	Jacobi	5.9375	600	8.7E-01	5.9375	1200
0.2500	ILU0	0.8438	101	9.8E-07	ILU0	1.2344	60	3.4E-03	ILU0	0.9375	60	3.6E-08	0.8438	101
0.2344	BlockILU	2.6875	131	9.2E-07	BlockILU	2.6406	53	4.8E-02	BlockILU	2.7031	70	2.6E-07	2.6875	131
0.2188	BlockJacobi	2.3125	131	9.2E-07	BlockJacobi	2.3594	53	4.6E-02	BlockJacobi	2.3906	70	2.6E-07	2.3125	131
0.2500	BlockJacobi2	2.5156	131	9.2E-07	BlockJacobi2	2.2188	53	4.6E-02	BlockJacobi2	14.7188	70	2.6E-07	2.5156	131
0.2188	GS	1.4219	83	8.6E-07	GS	2.2E+00	60	7.4E-04	GS	12.6094	40	5.6E-07	1.4219	80
0.2188	Multigrid	0.7344	27	5.6E-07	Multigrid	11.2188	0	2.2E+00	Multigrid	28.7500	10	4.7E-15	0.7344	20
													0.7344	**20**
dt=200		dt=200				dt=200				dt=200			dt=200	
0.2500	None	8.0625	1220	1.3E+00	None	0.5156	5	1.8E+01	None	5.1406	600	1.6E+00	5.1406	1200
0.2031	Upper	10.8125	1220	6.4E+24	Upper	0.0625	0	2.2E+01	Upper	121.1250	600		######	1200
0.1875	Lower	11.0000	1180	3.7E+23	Lower	0.9063	0	2.2E+01	Lower	122.0625	600		######	1200
0.2188	Jacobi	8.7656	1220	1.1E+00	Jacobi	0.6563	3	2.2E+01	Jacobi	5.9688	600	2.5E-01	5.9688	1200
0.2188	ILU0	5.3125	579	9.8E-07	ILU0	1.1406	54	2.2E+00	ILU0	2.4531	180	6.0E-07	2.4531	360
0.2031	BlockILU	16.4688	741	1.0E-06	BlockILU	2.6094	60	4.2E+00	BlockILU	11.3281	300	8.7E-07	11.3281	600
0.2500	BlockJacobi	14.5469	741	1.0E-06	BlockJacobi	2.3906	60	4.3E+00	BlockJacobi	9.8594	300	8.7E-07	9.8594	600
0.2031	BlockJacobi2	15.4688	741	1.0E-06	BlockJacobi2	2.2188	60	4.3E+00	BlockJacobi2	62.3125	300	8.7E-07	15.4688	600
0.2188	GS	7.3750	365	9.8E-07	GS	2.3E+00	57	2.8E+00	GS	49.8281	160	4.0E-07	7.3750	320
0.1875	Multigrid	0.8125	28	5.8E-07	Multigrid	11.2500	0	2.2E+01	Multigrid	28.9063	10	6.3E-13	0.8125	20
													0.8125	**20**
dt=2000		dt=2000				dt=2000				dt=2000			dt=2000	
0.2500	None	8.0625	1220	2.3E+01	None	0.4844	5	1.8E+02	None	5.7031	600	5.4E+01	5.7031	1200
0.1875	Upper	10.4219	720	3.7E+25	Upper	0.0938	0	2.2E+02	Upper	121.5469	600		######	1200
0.1875	Lower	11.4063	440	3.3E+24	Lower	0.9531	0	2.2E+02	Lower	122.5313	600		######	1200
0.1719	Jacobi	8.9219	1220	7.0E+00	Jacobi	0.6406	2	2.2E+02	Jacobi	5.6094	600	2.0E-01	5.6094	1200
0.2031	ILU0	11.7500	1220	2.3E-05	ILU0	1.2188	60	2.1E+01	ILU0	6.7344	550	9.1E-07	6.7344	1100
0.1875	BlockILU	28.2813	1220	3.7E-05	BlockILU	2.6094	60	5.1E+01	BlockILU	22.5625	600	6.3E-06	22.5625	1200
0.2344	BlockJacobi	23.3281	1220	3.7E-05	BlockJacobi	2.2969	60	5.3E+01	BlockJacobi	20.0000	600	6.3E-06	20.0000	1200
0.2031	BlockJacobi2	25.4375	1220	3.7E-05	BlockJacobi2	2.1563	60	5.3E+01	BlockJacobi2	124.2344	600	7.0E-06	######	1200
0.2188	GS	24.0625	1220	5.6E-06	GS	2.3E+00	60	1.8E+01	GS	155.2031	500	7.0E-07	######	1000
0.1875	Multigrid	0.9219	29	5.0E-07	Multigrid	11.2031	0	2.2E+02	Multigrid	29.1563	10	1.8E-11	0.9219	20
													0.9219	**20**

Table A.7: Primary Preconditioner/Solver benchmark results for $\kappa = 1, V_{scale} = 1$ using LDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

Slash	GmresR				Qmr				BiCgstab				Summary	
Time	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Precond.*	*Time*	*Iter.*	*Resid.*	*Min Time*	*Min MV*
dt=0.2		dt=0.2				dt=0.2				dt=0.2			dt=0.2	
0.5938	None	0.4688	66	9.6E-07	None	0.8438	50	7.3E-07	None	0.4063	30	1.2E-07	0.4063	60
0.6094	Upper	0.3438	42	7.5E-07	Upper	1.0313	32	6.1E-07	Upper	0.4219	20	8.2E-07	0.3438	40
0.5781	Lower	0.2969	37	9.3E-07	Lower	0.7031	26	3.9E-07	Lower	0.4063	20	1.2E-08	0.2969	37
0.5938	Jacobi	0.3750	49	7.3E-07	Jacobi	0.7344	43	9.5E-07	Jacobi	0.5313	30	2.5E-07	0.3750	49
0.5938	ILU0	0.1563	28	4.9E-07	ILU0	0.6094	10	5.1E-07	ILU0	0.2969	10	4.4E-11	0.1563	20
0.6094	BlockILU	0.5781	41	8.9E-07	BlockILU	1.4688	35	3.6E-07	BlockILU	1.0938	30	3.6E-09	0.5781	41
0.5938	BlockJacobi	0.5938	41	8.9E-07	BlockJacobi	1.6094	35	3.1E-07	BlockJacobi	1.1094	30	3.6E-09	0.5938	41
0.6094	BlockJacobi2	1.0625	41	8.9E-07	BlockJacobi2	2.5000	35	3.1E-07	BlockJacobi2	9.4844	30	3.6E-09	1.0625	41
0.5938	GS	0.7344	33	7.3E-07	GS	1.92188	19	7.7E-07	GS	19.84375	20	3.7E-10	0.7344	33
0.6094	Multigrid	1.0781	28	8.7E-07	Multigrid	10.9219	43	1.4E-01	Multigrid	29.9063	10	1.0E-12	1.0781	20
													0.1563	20
dt=2		dt=2				dt=2				dt=2			dt=2	
0.7031	None	3.1094	343	9.7E-07	None	0.8594	60	4.0E-02	None	1.9688	170	8.0E-07	1.9688	340
0.6406	Upper	2.5938	206	9.9E-07	Upper	1.8906	31	4.5E-02	Upper	2.2188	130	1.7E-07	2.2188	206
0.6094	Lower	1.1563	105	9.5E-07	Lower	1.6250	60	5.4E-03	Lower	1.2500	80	9.2E-07	1.1563	105
0.6719	Jacobi	2.2188	254	1.0E-06	Jacobi	0.9688	54	5.8E-02	Jacobi	1.9375	160	8.9E-07	1.9375	254
0.6250	ILU0	18.0156	1220	2.4E-04	ILU0	2.5156	60	7.0E-01	ILU0	10.9531	600	1.1E-05	10.9531	1200
0.6250	BlockILU	3.0156	133	9.9E-07	BlockILU	2.8594	60	3.0E-02	BlockILU	3.9688	100	6.1E-07	3.0156	133
0.6406	BlockJacobi	2.6563	133	9.9E-07	BlockJacobi	2.5313	60	2.7E-02	BlockJacobi	3.6406	100	6.1E-07	2.6563	133
0.6250	BlockJacobi2	4.2656	133	9.9E-07	BlockJacobi2	4.1875	60	2.7E-02	BlockJacobi2	30.9219	100	7.2E-07	4.2656	133
0.6094	GS	3.0000	80	8.9E-07	GS	5.34375	58	9.3E-04	GS	58.375	60	4.4E-07	3.0000	80
0.6563	Multigrid	1.4375	31	9.7E-07	Multigrid	15.2969	0	1.5E+00	Multigrid	29.9219	10	3.4E-10	1.4375	20
													1.1563	20
dt=20		dt=20				dt=20				dt=20			dt=20	
0.6563	None	10.7344	1220	1.8E-03	None	0.7188	38	1.1E+00	None	6.4531	600	6.7E-04	6.4531	1200
0.6250	Upper	9.9688	788	1.0E-06	Upper	1.9219	52	1.8E+00	Upper	6.7656	420	3.7E-07	6.7656	788
0.6250	Lower	5.4688	469	1.0E-06	Lower	1.5000	9	3.7E+00	Lower	3.8594	260	9.7E-07	3.8594	469
0.6875	Jacobi	8.2656	906	1.0E-06	Jacobi	0.9688	26	1.7E+00	Jacobi	6.4688	520	8.0E-07	6.4688	906
0.6250	ILU0	18.0156	1220	7.7E-03	ILU0	2.3750	45	9.1E+00	ILU0	10.9531	600	2.0E-02	10.9531	1200
0.6250	BlockILU	13.0469	551	1.0E-06	BlockILU	2.9063	37	1.2E+00	BlockILU	16.7813	430	1.4E-07	13.0469	551
0.6406	BlockJacobi	11.5469	551	1.0E-06	BlockJacobi	2.6250	40	1.3E+00	BlockJacobi	15.0469	430	1.4E-07	11.5469	551
0.6250	BlockJacobi2	19.7188	551	1.0E-06	BlockJacobi2	4.2031	40	1.3E+00	BlockJacobi2	126.3750	410	9.6E-07	19.7188	551
0.6250	GS	13.9688	317	1.0E-06	GS	5.3E+00	59	2.2E+00	GS	219.0156	230	2.3E-07	13.9688	317
0.6250	Multigrid	1.6719	33	6.8E-07	Multigrid	15.3281	0	1.5E+01	Multigrid	30.3281	10	5.7E-09	1.6719	20
													1.6719	20
dt=200		dt=200				dt=200				dt=200			dt=200	
0.7031	None	10.6719	1220	7.0E-01	None	0.8438	38	1.1E+01	None	6.8594	600	2.2E+00	6.8594	1200
0.5938	Upper	15.8125	1220	3.1E-05	Upper	1.8594	47	2.3E+01	Upper	9.8906	600	2.6E-05	9.8906	1200
0.6250	Lower	14.6719	1220	1.6E-05	Lower	1.5625	9	3.6E+01	Lower	8.6250	600	3.3E-06	8.6250	1200
0.6719	Jacobi	11.0938	1220	3.9E-05	Jacobi	0.8125	26	1.9E+01	Jacobi	7.2344	600	3.1E-05	7.2344	1200
0.6406	ILU0	18.0625	1220	2.3E-02	ILU0	2.5313	52	1.2E+02	ILU0	11.0156	600	2.0E-01	11.0156	1200
0.6406	BlockILU	29.5781	1220	5.0E-06	BlockILU	2.8594	37	2.0E+01	BlockILU	23.3125	600	1.1E-05	23.3125	1200
0.6250	BlockJacobi	26.5156	1220	5.0E-06	BlockJacobi	2.6094	37	1.9E+01	BlockJacobi	21.0156	600	1.1E-05	21.0156	1200
0.6250	BlockJacobi2	44.3125	1220	5.0E-06	BlockJacobi2	4.2188	37	1.9E+01	BlockJacobi2	184.6094	600	1.2E-05	######	1200
0.6250	GS	56.1406	1220	1.9E-06	GS	5.4E+00	17	5.1E+01	GS	398.8594	420	5.9E-07	######	840
0.6563	Multigrid	1.7656	34	4.2E-07	Multigrid	15.3594	0	1.5E+02	Multigrid	30.4688	10	1.6E-09	1.7656	20
													1.7656	20
dt=2000		dt=2000				dt=2000				dt=2000			dt=2000	
0.6875	None	10.7031	1220	1.0E+01	None	0.9375	37	1.1E+02	None	6.5625	600	1.9E+01	6.5625	1200
0.6406	Upper	15.7500	1220	7.4E-05	Upper	1.9531	55	2.4E+02	Upper	9.8594	600	1.2E-04	9.8594	1200
0.6250	Lower	14.6406	1220	5.5E-05	Lower	1.5313	9	3.6E+02	Lower	8.6563	600	2.5E-05	8.6563	1200
0.6563	Jacobi	11.5000	1220	7.5E-05	Jacobi	0.8594	26	1.9E+02	Jacobi	7.2031	600	9.4E-05	7.2031	1200
0.6250	ILU0	18.2188	1220	3.8E-02	ILU0	2.5156	46	1.2E+03	ILU0	10.6875	600	3.6E-03	10.6875	1200
0.6250	BlockILU	30.3438	1220	1.1E-05	BlockILU	2.8594	37	2.1E+02	BlockILU	23.6875	600	1.3E-05	23.6875	1200
0.6250	BlockJacobi	26.3750	1220	1.1E-05	BlockJacobi	2.6563	37	2.0E+02	BlockJacobi	20.9063	600	1.3E-05	20.9063	1200
0.6719	BlockJacobi2	44.7344	1220	1.1E-05	BlockJacobi2	4.2188	37	2.0E+02	BlockJacobi2	184.8125	600	1.1E-05	######	1200
0.6250	S	55.8906	1220	1.2E-05	GS	5.3E+00	17	5.1E+02	GS	576.6563	600	1.4E-06	######	1200
0.6250	Multigrid	1.7656	34	4.9E-07	Multigrid	15.2656	0	1.5E+03	Multigrid	31.5313	10	2.1E-09	1.7656	20
													1.7656	20

Table A.8: GMRES versus BiCGSTAB(l) restart benchmark results for $\kappa = 1, V_{scale} = 0$ using LDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

GmresR					BiCgstab					Summary	
Re-starts	Time	Iter.	Resid.	Flag	Re-starts	Time	Iter.	Resid.	Flag	Min Time	Min MV
dt=0.2					dt=0.2					dt=0.2	
2	0.1875	10	4.5E-07	0	1	0.2344	6	9.8E-07	0	0.1875	10
4	0.1875	11	8.4E-07	0	2	0.2500	6	5.6E-07	0	0.1875	11
8	0.1719	15	7.5E-07	0	4	0.2813	8	1.6E-08	0	0.1719	15
10	0.1719	17	7.5E-07	0	5	0.3125	10	3.5E-11	0	0.1719	17
12	0.2031	19	7.5E-07	0	6	0.2656	6	2.4E-07	0	0.2031	12
14	0.1719	21	7.5E-07	0	7	0.2813	7	1.1E-07	0	0.1719	14
16	0.1719	23	7.5E-07	0	8	0.3125	8	1.0E-08	0	0.1719	16
18	0.2031	25	7.5E-07	0	9	0.3438	9	2.6E-10	0	0.2031	18
20	0.1719	27	7.5E-07	0	10	0.3438	10	2.0E-11	0	0.1719	20
24	0.1719	31	7.5E-07	0	12	0.4375	12	4.6E-13	0	0.1719	24
30	0.1719	37	7.5E-07	0	15	0.5156	15	7.6E-13	0	0.1719	30
40	0.1719	47	7.5E-07	0	20	0.7031	20	2.9E-14	0	0.1719	40
50	0.1719	57	7.5E-07	0	25	0.9531	25	1.7E-14	0	0.1719	50
										0.1719	10
dt=2					dt=2					dt=2	
2	19.5313	1200	8.8E-03	1	1	4.6563	297	9.7E-07	0	4.6563	594
4	17.2969	652	1.0E-02	1	2	5.0625	348	4.0E-07	0	5.0625	696
8	16.3750	1208	2.2E-03	1	4	5.5313	360	3.2E-07	0	5.5313	720
10	16.7031	1210	3.1E-04	1	5	5.7344	360	2.4E-07	0	5.7344	720
12	16.7500	1212	1.1E-04	1	6	6.0625	366	7.0E-07	0	6.0625	732
14	17.1406	1218	2.0E-05	1	7	6.1719	364	9.9E-07	0	6.1719	728
16	17.6406	1216	2.4E-05	1	8	6.0469	360	6.9E-07	0	6.0469	720
18	18.2188	1224	1.6E-05	1	9	6.9219	378	2.0E-07	0	6.9219	756
20	18.4375	1220	4.4E-06	1	10	6.9688	370	4.5E-07	0	6.9688	740
24	19.4375	1224	1.7E-06	1	12	7.4531	372	4.9E-07	0	7.4531	744
30	13.1719	785	9.9E-07	0	15	9.0313	405	7.1E-07	0	9.0313	785
40	9.5938	530	1.0E-06	0	20	13.9844	460	8.2E-07	0	9.5938	530
50	8.9375	438	9.8E-07	0	25	20.2656	600	1.0E-03	1	8.9375	438
										4.6563	438
dt=20					dt=20					dt=20	
2	19.6875	1202	1.1E-01	1	1	8.9688	600	9.1E-03	1	#######	#####
4	17.0156	1204	5.1E-02	1	2	8.7656	600	8.7E-03	1	#######	#####
8	16.4531	416	7.7E-03	1	4	9.0156	600	1.0E-02	1	#######	#####
10	16.6094	1210	6.1E-03	1	5	8.9531	600	3.0E-02	1	#######	#####
12	17.1406	1212	5.9E-03	1	6	9.7656	600	8.5E-03	1	#######	#####
14	16.8906	1218	5.8E-03	1	7	10.0625	602	1.0E-02	1	#######	#####
16	17.7031	1216	5.5E-03	1	8	10.3750	600	1.0E-02	1	#######	#####
18	18.6094	1224	4.8E-03	1	9	10.9688	603	1.9E-02	1	#######	#####
20	18.4688	1220	3.9E-03	1	10	10.9063	600	1.1E-02	1	#######	#####
24	19.5625	1224	3.3E-03	1	12	11.9531	600	8.4E-03	1	#######	#####
30	20.4531	1230	2.4E-03	1	15	12.9375	600	9.7E-03	1	#######	#####
40	23.3906	1240	2.4E-03	1	20	18.9844	600	1.2E-02	1	#######	#####
50	25.3906	1250	2.2E-03	1	25	20.3281	600	8.1E+01	1	#######	#####
										#######	#####
dt=200					dt=200					dt=200	
2	19.6250	1202	2.3E-02	1	1	8.9063	600	1.2E-02	1	#######	#####
4	17.2031	1204	7.5E-03	1	2	8.6875	600	4.9E-03	1	#######	#####
8	16.4844	1208	3.5E-03	1	4	9.0313	600	2.7E-03	1	#######	#####
10	16.8750	1210	3.5E-03	1	5	8.8750	600	4.0E-03	1	#######	#####
12	17.1094	1212	3.1E-03	1	6	9.2813	600	4.1E-03	1	#######	#####
14	17.4375	1218	2.9E-03	1	7	9.9219	602	7.0E-03	1	#######	#####
16	17.4531	1216	2.8E-03	1	8	10.3906	600	4.7E-03	1	#######	#####
18	18.3594	1224	2.7E-03	1	9	10.9219	603	3.2E-03	1	#######	#####
20	18.9063	1220	2.7E-03	1	10	10.6094	600	1.0E-01	1	#######	#####
24	19.7500	1224	2.6E-03	1	12	11.3438	600	2.0E-03	1	#######	#####
30	20.4063	1230	2.5E-03	1	15	12.3750	600	2.4E-03	1	#######	#####
40	22.7344	1240	2.3E-03	1	20	19.0313	600	5.8E-03	1	#######	#####
50	25.6094	1250	2.0E-03	1	25	20.2188	600	8.2E-03	1	#######	#####
										#######	#####
dt=2000					dt=2000					dt=2000	
2	19.5781	1166	1.4E-01	1	1	8.8594	600	3.4E-04	1	#######	#####
4	16.9063	1204	8.0E-03	1	2	8.7188	600	4.2E-04	1	#######	#####
8	16.3906	1208	7.9E-03	1	4	9.1094	600	3.4E-04	1	#######	#####
10	16.6406	1210	3.5E-03	1	5	9.2813	600	3.2E-04	1	#######	#####
12	16.6719	1212	4.2E-03	1	6	9.2813	600	3.6E-04	1	#######	#####
14	17.0469	1218	2.8E-03	1	7	10.1719	602	4.4E-04	1	#######	#####
16	17.6094	1216	2.8E-03	1	8	10.7031	600	3.4E-04	1	#######	#####
18	18.4063	1224	2.6E-03	1	9	10.4219	603	2.3E-04	1	#######	#####
20	18.2188	1220	2.6E-03	1	10	11.1250	600	2.4E-04	1	#######	#####
24	19.4531	1224	2.6E-03	1	12	11.3438	600	2.8E-04	1	#######	#####
30	20.6250	1230	2.6E-03	1	15	12.8906	600	4.1E-04	1	#######	#####
40	23.4375	1240	2.4E-03	1	20	18.6094	600	8.6E-04	1	#######	#####
50	26.4375	1250	2.3E-03	1	25	20.0469	600	1.4E-02	1	#######	#####
										#######	#####

Table A.9: GMRES versus BiCGSTAB(l) restart benchmark results for $\kappa = 0, V_{scale} = 1$ using LDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

GmresR					BiCgstab					Summary	
Re-starts	Time	Iter.	Resid.	Flag	Re-starts	Time	Iter.	Resid.	Flag	Min Time	Min MV
dr=0.2					dr=0.2					dr=0.2	
2	0.1719	6	8.3E-07	0	1	0.3281	3	7.3E-07	0	0.1719	6
4	0.1094	8	6.6E-07	0	2	0.2031	4	6.4E-10	0	0.1094	8
8	0.1250	12	6.6E-07	0	4	0.1719	4	5.2E-11	0	0.1250	8
10	0.0938	14	6.6E-07	0	5	0.1875	5	5.0E-13	0	0.0938	10
12	0.1250	16	6.6E-07	0	6	0.2344	6	9.0E-15	0	0.1250	12
14	0.1250	18	6.6E-07	0	7	0.2344	7	1.5E-16	0	0.1250	14
16	0.1250	20	6.6E-07	0	8	0.2969	8	1.0E-14	0	0.1250	16
18	0.0938	22	6.6E-07	0	9	0.2188	9	2.3E-13	0	0.0938	18
20	0.1250	24	6.6E-07	0	10	0.2344	10	4.4E-14	0	0.1250	20
24	0.1250	28	6.6E-07	0	12	0.3594	12	2.0E-14	0	0.1250	24
30	0.1250	34	6.6E-07	0	15	0.4219	15	4.5E-16	0	0.1250	30
40	0.1250	44	6.6E-07	0	20	0.5625	20	9.0E-17	0	0.1250	40
50	0.1406	54	6.6E-07	0	25	0.8750	25	1.1E-16	0	0.1406	50
										0.0938	6
dt=2					dt=2					dt=2	
2	0.1094	14	9.8E-07	0	1	0.2344	8	1.1E-07	0	0.1094	14
4	0.1563	15	8.1E-07	0	2	0.2188	8	1.4E-07	0	0.1563	15
8	0.1719	19	8.2E-07	0	4	0.2500	8	9.0E-08	0	0.1719	16
10	0.1406	21	8.1E-07	0	5	0.2969	10	2.8E-09	0	0.1406	20
12	0.2031	23	7.9E-07	0	6	0.2500	12	4.1E-12	0	0.2031	23
14	0.2188	25	7.9E-07	0	7	0.2656	7	5.7E-07	0	0.2188	14
16	0.1563	27	7.9E-07	0	8	0.2188	8	5.6E-08	0	0.1563	16
18	0.2500	29	7.9E-07	0	9	0.2188	9	2.0E-08	0	0.2188	18
20	0.2188	31	7.9E-07	0	10	0.3125	10	3.4E-08	0	0.2188	20
24	0.1875	35	7.9E-07	0	12	0.3438	12	3.0E-10	0	0.1875	24
30	0.1719	41	7.9E-07	0	15	0.3906	15	1.2E-10	0	0.1719	30
40	0.1875	51	7.9E-07	0	20	0.5313	20	6.6E-11	0	0.1875	40
50	0.1719	61	7.9E-07	0	25	0.6719	25	3.3E-11	0	0.1719	50
										0.1094	14
dt=20					dt=20					dt=20	
2	0.9063	92	8.8E-07	0	1	0.7656	59	4.8E-07	0	0.7656	92
4	0.8750	93	9.6E-07	0	2	0.5781	50	6.9E-07	0	0.5781	93
8	0.8125	96	9.2E-07	0	4	0.5469	48	8.0E-07	0	0.5469	96
10	0.8906	96	8.9E-07	0	5	0.6406	50	8.4E-07	0	0.6406	96
12	0.9063	96	9.6E-07	0	6	0.6563	48	7.9E-07	0	0.6563	96
14	0.8125	97	9.2E-07	0	7	0.6719	49	8.0E-07	0	0.6719	97
16	1.0000	101	9.1E-07	0	8	0.6250	48	7.9E-07	0	0.6250	96
18	0.9688	102	9.3E-07	0	9	0.8125	54	6.6E-07	0	0.8125	102
20	0.9531	101	9.8E-07	0	10	0.9688	60	3.6E-08	0	0.9531	101
24	0.9219	105	8.9E-07	0	12	0.8906	48	9.0E-07	0	0.8906	96
30	0.9219	107	9.4E-07	0	15	1.0781	60	4.2E-08	0	0.9219	107
40	1.2344	115	9.0E-07	0	20	1.6563	60	3.7E-07	0	1.2344	115
50	1.2188	126	9.1E-07	0	25	2.0625	75	6.8E-08	0	1.2188	126
										0.5469	92
dt=200					dt=200					dt=200	
2	5.6406	662	9.8E-07	0	1	2.5000	267	6.4E-07	0	2.5000	534
4	4.8594	633	1.0E-06	0	2	1.8594	192	8.4E-07	0	1.8594	384
8	5.0313	634	9.9E-07	0	4	1.8594	184	8.9E-07	0	1.8594	368
10	5.1719	636	1.0E-06	0	5	1.8125	180	8.4E-07	0	1.8125	360
12	5.1250	604	1.0E-06	0	6	1.8438	180	5.8E-07	0	1.8438	360
14	5.0625	582	9.8E-07	0	7	2.1875	182	5.6E-07	0	2.1875	364
16	5.4844	582	1.0E-06	0	8	2.1563	184	4.8E-07	0	2.1563	368
18	5.2656	555	1.0E-06	0	9	2.2969	180	7.0E-07	0	2.2969	360
20	5.5156	579	9.8E-07	0	10	2.3438	180	6.0E-07	0	2.3438	360
24	5.1875	499	1.0E-06	0	12	2.7656	180	5.6E-07	0	2.7656	360
30	5.6719	521	9.9E-07	0	15	2.9844	180	9.8E-07	0	2.9844	360
40	5.3906	422	9.9E-07	0	20	4.9688	200	3.1E-07	0	4.9688	400
50	5.6094	381	9.9E-07	0	25	6.4531	250	4.2E-07	0	5.6094	381
										1.8125	360
dt=2000					dt=2000					dt=2000	
2	10.4688	1170	1.2E-04	1	1	5.3750	581	6.5E-07	0	5.3750	1162
4	9.1406	1204	3.9E-05	1	2	4.7656	528	9.0E-07	0	4.7656	1056
8	9.8281	1208	3.3E-05	1	4	5.0313	540	8.1E-07	0	5.0313	1080
10	9.8906	1210	3.2E-05	1	5	5.3281	540	8.1E-07	0	5.3281	1080
12	10.2500	1212	2.8E-05	1	6	5.6406	534	8.7E-07	0	5.6406	1068
14	10.3750	1218	2.4E-05	1	7	6.0000	539	4.9E-07	0	6.0000	1078
16	11.0938	1216	2.5E-05	1	8	6.7656	560	3.6E-07	0	6.7656	1120
18	11.6250	1224	2.6E-05	1	9	6.6406	558	7.4E-07	0	6.6406	1116
20	11.7500	1220	2.3E-05	1	10	7.1875	550	9.1E-07	0	7.1875	1100
24	13.2969	1224	1.5E-05	1	12	7.0000	504	4.5E-07	0	7.0000	1008
30	15.2344	1230	1.8E-05	1	15	8.9219	555	8.9E-07	0	8.9219	1110
40	17.5938	1240	3.4E-06	1	20	15.7188	600	6.7E-05	1	#######	#####
50	19.1563	1250	2.6E-06	1	25	16.0313	600	6.5E-06	1	#######	#####
										4.7656	1008

Table A.10: GMRES versus BiCGSTAB(l) restart benchmark results for $\kappa = 1, V_{scale} = 1$ using LDG discretization. Red highlighting indicates the solution did not converge. The fastest simulation for a given CFL number is highlighted in green, and the iteration with the fewest matrix-vector multiplications is outlined in orange.

GmresR					BiCgstab					Summary	
Re-starts	Time	Iter.	Resid.	Flag	Re-starts	Time	Iter.	Resid.	Flag	Min Time	Min MV
dt=0.2					dt=0.2					dt=0.2	
2	0.2031	10	1.0E-06	0	1	0.2813	7	2.2E-07	0	0.2031	10
4	0.2188	12	5.5E-07	0	2	0.3125	8	2.1E-08	0	0.2188	12
8	0.2031	16	4.9E-07	0	4	0.2969	8	1.7E-08	0	0.2031	16
10	0.1875	18	4.9E-07	0	5	0.3594	10	6.9E-11	0	0.1875	18
12	0.2031	20	4.9E-07	0	6	0.3125	6	7.6E-07	0	0.2031	12
14	0.1875	22	4.9E-07	0	7	0.2813	7	7.5E-08	0	0.1875	14
16	0.1719	24	4.9E-07	0	8	0.3125	8	1.4E-08	0	0.1719	16
18	0.1875	26	4.9E-07	0	9	0.3281	9	5.9E-10	0	0.1875	18
20	0.1875	28	4.9E-07	0	10	0.3438	10	4.4E-11	0	0.1875	20
24	0.1875	32	4.9E-07	0	12	0.3750	12	9.3E-12	0	0.1875	24
30	0.1875	38	4.9E-07	0	15	0.5313	15	5.0E-12	0	0.1875	30
40	0.2031	48	4.9E-07	0	20	0.7500	20	6.3E-14	0	0.2031	40
50	0.2031	58	4.9E-07	0	25	0.8594	25	3.9E-11	0	0.2031	50
										0.1719	10
dt=2					dt=2					dt=2	
2	21.3594	1202	1.0E-01	1	1	9.6563	600	6.1E-05	1	#######	#####
4	17.1094	1204	4.7E-02	1	2	8.8594	600	3.7E-05	1	#######	#####
8	16.9375	1208	2.3E-02	1	4	9.1406	600	2.0E-05	1	#######	#####
10	16.9531	1210	1.5E-02	1	5	9.4688	600	1.3E-05	1	#######	#####
12	17.1719	1212	9.2E-03	1	6	9.6250	600	9.2E-06	1	#######	#####
14	17.1719	1218	4.6E-03	1	7	10.0781	602	6.2E-06	1	#######	#####
16	18.0156	1216	5.7E-04	1	8	10.3125	600	7.0E-06	1	#######	#####
18	18.4375	1224	6.6E-04	1	9	10.8125	603	1.3E-05	1	#######	#####
20	18.5938	1220	2.4E-04	1	10	11.1719	600	1.1E-05	1	#######	#####
24	19.3281	1224	7.7E-05	1	12	11.8438	600	3.0E-05	1	#######	#####
30	21.3281	1230	1.1E-05	1	15	12.9375	600	3.9E-05	1	#######	#####
40	17.4844	906	9.9E-07	0	20	19.1250	600	1.3E-04	1	17.4844	906
50	9.6250	496	9.9E-07	0	25	19.4531	600	6.6E+00	1	9.6250	496
										9.6250	496
dt=20					dt=20					dt=20	
2	19.4531	762	4.1E-02	1	1	8.8906	600	1.2E-02	1	#######	#####
4	17.0000	1204	4.9E-02	1	2	8.6563	600	1.3E-02	1	#######	#####
8	16.3281	1208	3.4E-02	1	4	8.2813	600	1.2E-02	1	#######	#####
10	16.7031	1210	2.4E-02	1	5	9.3906	600	1.8E-02	1	#######	#####
12	17.1875	1212	2.0E-02	1	6	9.2344	600	1.2E-02	1	#######	#####
14	17.4688	1218	1.4E-02	1	7	9.6875	602	5.1E-02	1	#######	#####
16	18.0625	1216	1.2E-02	1	8	10.2656	600	1.2E-02	1	#######	#####
18	18.0469	1224	1.1E-02	1	9	10.7031	603	5.2E-02	1	#######	#####
20	18.4219	1220	7.7E-03	1	10	11.0938	600	2.0E-02	1	#######	#####
24	19.0625	1224	6.4E-03	1	12	11.7813	600	1.9E-02	1	#######	#####
30	21.0625	1230	5.3E-03	1	15	12.4375	600	1.8E-02	1	#######	#####
40	23.7500	1240	7.7E-04	1	20	19.0000	600	1.9E-02	1	#######	#####
50	26.3125	1250	5.2E-04	1	25	19.1719	600	8.6E+00	1	#######	#####
										#######	#####
dt=200					dt=200					dt=200	
2	19.6719	1202	5.1E-02	1	1	8.9844	600	1.1E-02	1	#######	#####
4	17.0000	240	5.4E-02	1	2	8.6406	600	1.9E-02	1	#######	#####
8	16.4531	1208	3.5E-02	1	4	9.1875	600	3.1E-02	1	#######	#####
10	16.6406	1210	3.7E-02	1	5	9.3281	600	8.7E-03	1	#######	#####
12	16.8125	1212	3.4E-02	1	6	9.3906	600	8.4E-03	1	#######	#####
14	17.4375	1218	4.2E-02	1	7	10.0625	602	1.3E-02	1	#######	#####
16	17.9531	1216	2.4E-02	1	8	10.4219	600	8.7E-02	1	#######	#####
18	18.2969	1224	3.9E-02	1	9	10.7969	603	3.1E-02	1	#######	#####
20	18.6563	1220	2.3E-02	1	10	11.1563	600	2.0E-01	1	#######	#####
24	19.2188	1224	1.7E-02	1	12	11.9531	600	1.2E-02	1	#######	#####
30	21.3750	1230	1.4E-02	1	15	12.3438	600	2.4E-02	1	#######	#####
40	22.6563	1240	9.9E-03	1	20	19.0781	600	2.4E-02	1	#######	#####
50	27.0156	1250	6.7E-03	1	25	20.0781	600	1.2E+00	1	#######	#####
										#######	#####
dt=2000					dt=2000					dt=2000	
2	19.7031	748	5.4E-02	1	1	8.9531	600	9.0E-03	1	#######	#####
4	16.9531	1112	4.5E-02	1	2	8.7031	600	4.9E-03	1	#######	#####
8	16.4375	1208	3.9E-02	1	4	9.1719	600	3.5E-03	1	#######	#####
10	16.6406	1210	4.0E-02	1	5	8.8750	600	1.3E-02	1	#######	#####
12	17.1250	1212	3.9E-02	1	6	9.2656	600	3.0E-03	1	#######	#####
14	17.3906	1218	3.9E-02	1	7	10.1719	602	3.1E-03	1	#######	#####
16	17.5156	1216	4.0E-02	1	8	10.5156	600	3.3E-03	1	#######	#####
18	18.5000	1224	3.8E-02	1	9	10.3906	603	2.9E-03	1	#######	#####
20	18.8438	1220	3.8E-02	1	10	10.7969	600	3.6E-03	1	#######	#####
24	19.7813	1224	3.7E-02	1	12	11.7500	600	8.6E-03	1	#######	#####
30	20.7344	1230	3.4E-02	1	15	13.0000	600	3.2E-03	1	#######	#####
40	23.5469	1240	2.7E-02	1	20	18.5000	600	1.6E-03	1	#######	#####
50	26.6406	1250	1.6E-02	1	25	20.0313	600	1.6E-03	1	#######	#####
										#######	#####

Appendix B

Description of MATLAB functions/scripts

B.1 Functions and Helper Scripts for Implicit Integration

The following is a list including a short description of some of the relevant scripts/functions written for this study.

- `Bench.m`: This is the main program function. It accepts the various solver or problem parameters and outputs the computation time, error flag, residual, iterations till convergence, and the problem matrix. It is set up to do multiple integrations (advancing the solution in time) for multiple constituents. It either computes or loads from a file the preconditioner and problem matrix. Using multiple switch statements it chooses between different solvers and preconditioners. The computation time is taken from the onset of the time integration till its completion, and does not include the time to calculate the preconditioner or problem matrix. The rational behind this is that the formation of the matrix/preconditioners used in this implementation would not be representative of the computational time of an optimized implementation, and can therefore not be compared (in terms of time) to a MATLAB-implemented preconditioner

such as ILU(0).

- `BenchDriver.m`: This is a driver script containing multiple loops to make calls to `Bench.m` with various input parameters. There are multiple versions for the driver script for the different numerical experiments, but each have essentially the same structure with slight differences.

- `BenchXcelOutput.m`: Script to convert MATLAB output to Excel for ease of analysis.

- `Preconditioner.m`: This function implements some of the preconditioners, and computes sparse matrix preconditioners for use in the solvers.

- `BlockPreconditioner.m`: This function implements some of the preconditioners, and is passed as a function for use in the solver, returning the product $\mathbf{M}^{-1}x$.

- `MG.m`: Script used to implement a more advanced MG preconditioner with an ILU(0) preconditioner.

Appendix C

Figures

C.1 Convergence plots for Multigrid Benchmark

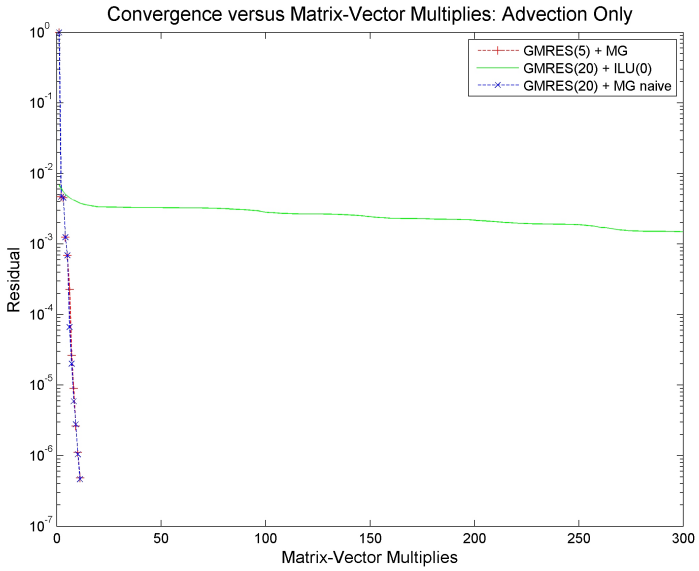

Figure C-1: Convergence history of GMRES(m) solver using different preconditioners. Here the naive MG preconditioner is used to precondition the GMRES(m) smoother for a pure advection case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

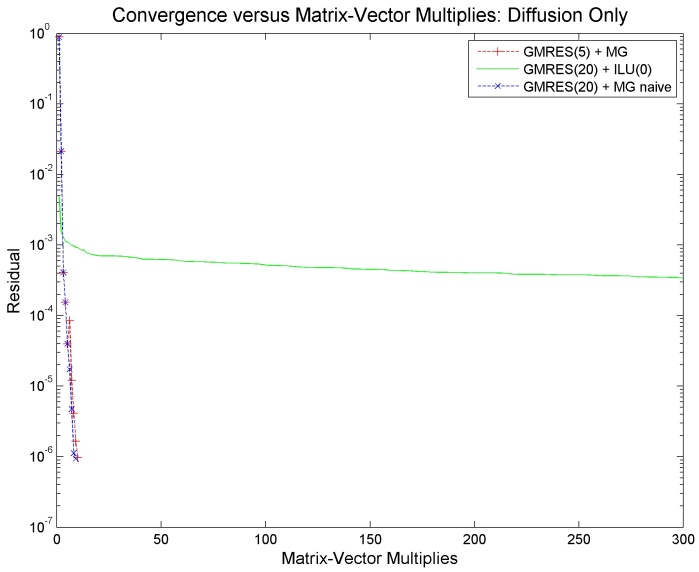

Figure C-2: Convergence history of GMRES(m) solver using different preconditioners. Here the naive MG preconditioner is used to precondition the GMRES(m) smoother for a pure diffusion case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

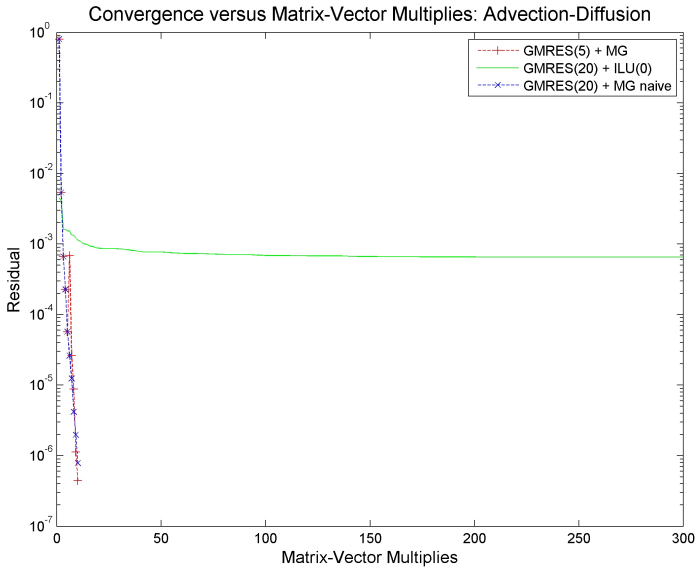

Figure C-3: Convergence history of GMRES(m) solver using different preconditioners. Here the naive MG preconditioner is used to precondition the GMRES(m) smoother for advection-diffusion case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

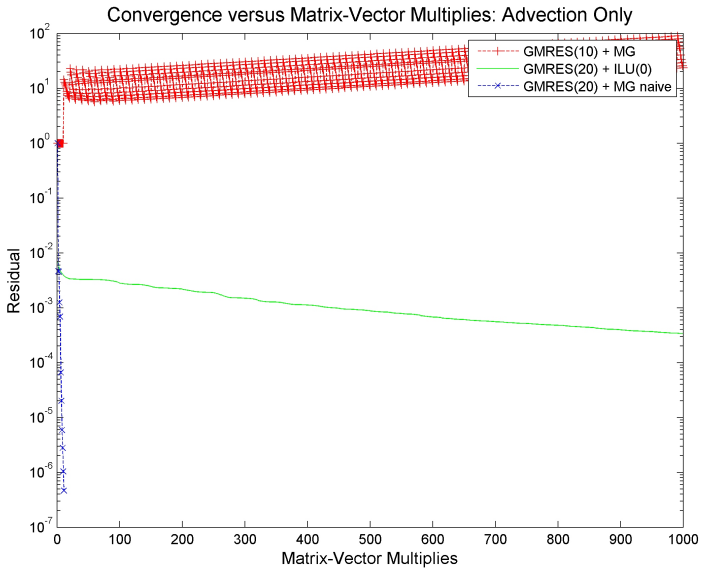

Figure C-4: Convergence history of GMRES(m) solver using different preconditioners. Here no preconditioner is used to precondition the GMRES(m) smoother for a pure advection case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

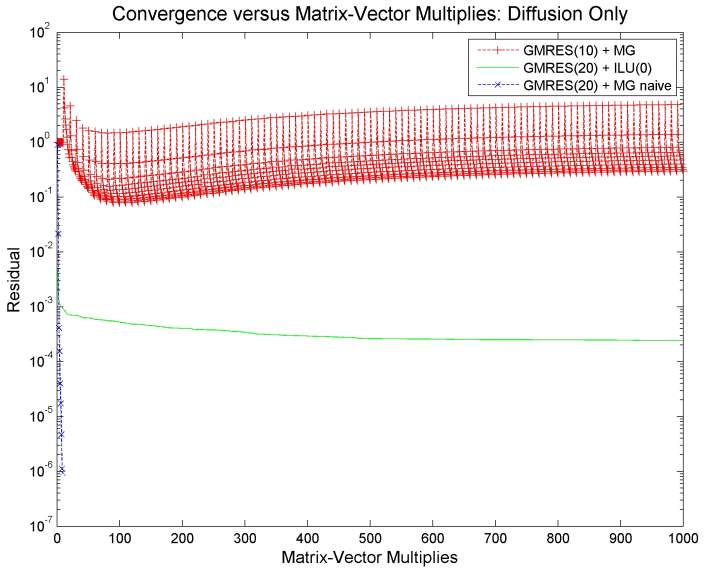

Figure C-5: Convergence history of GMRES(m) solver using different preconditioners. Here no preconditioner is used to precondition the GMRES(m) smoother for a pure diffusion case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

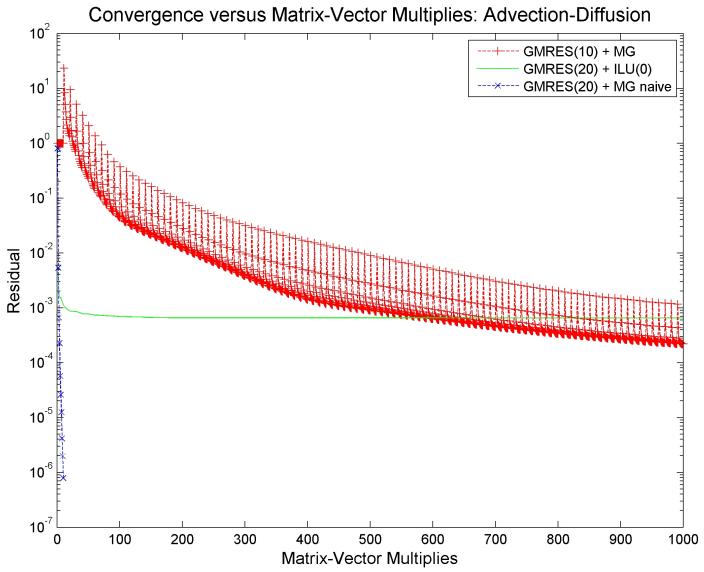

Figure C-6: Convergence history of GMRES(m) solver using different preconditioners. Here no preconditioner is used to precondition the GMRES(m) smoother for advection-diffusion case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

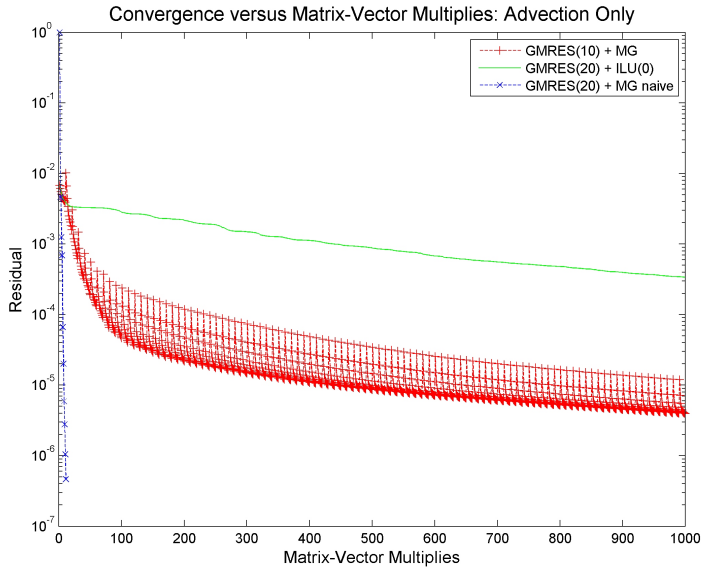

Figure C-7: Convergence history of GMRES(m) solver using different preconditioners. Here the ILU(0) preconditioner is used to precondition the GMRES(m) smoother for a pure advection case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

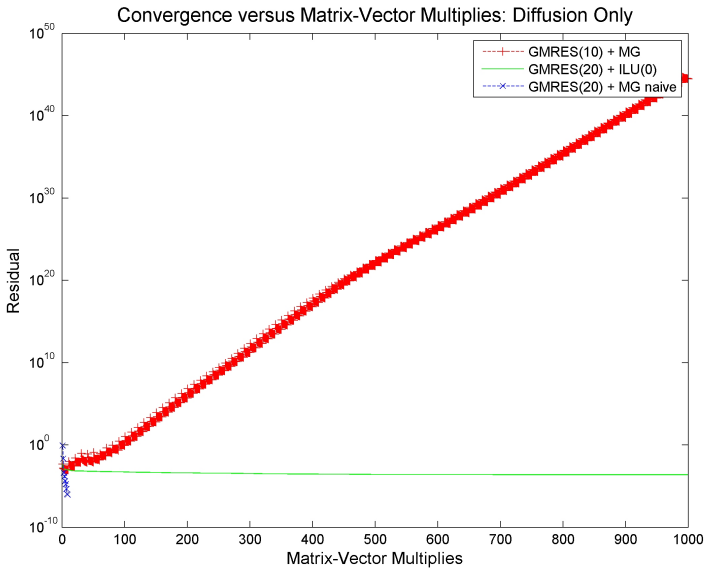

Figure C-8: Convergence history of GMRES(m) solver using different preconditioners. Here the ILU(0) preconditioner is used to precondition the GMRES(m) smoother for a pure diffusion case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

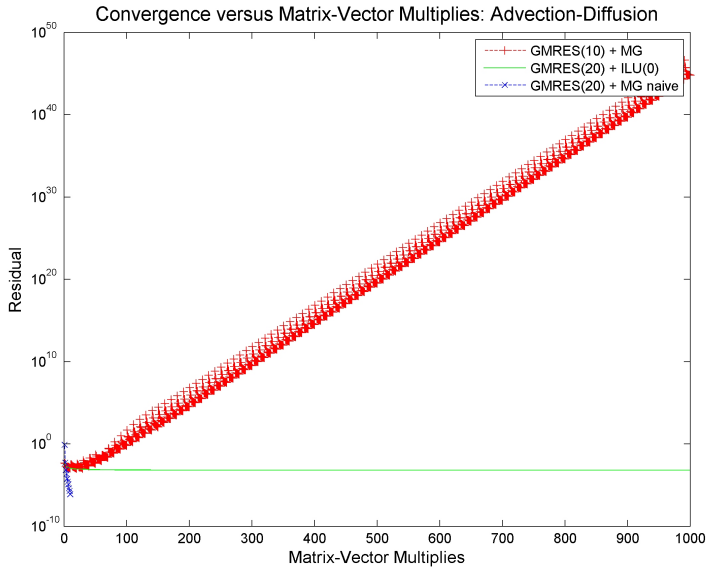

Figure C-9: Convergence history of GMRES(m) solver using different preconditioners. Here ILU(0) preconditioner is used to precondition the GMRES(m) smoother for advection-diffusion case with a proper p-MG implementation. Fourth order basis functions are used on the fine grid.

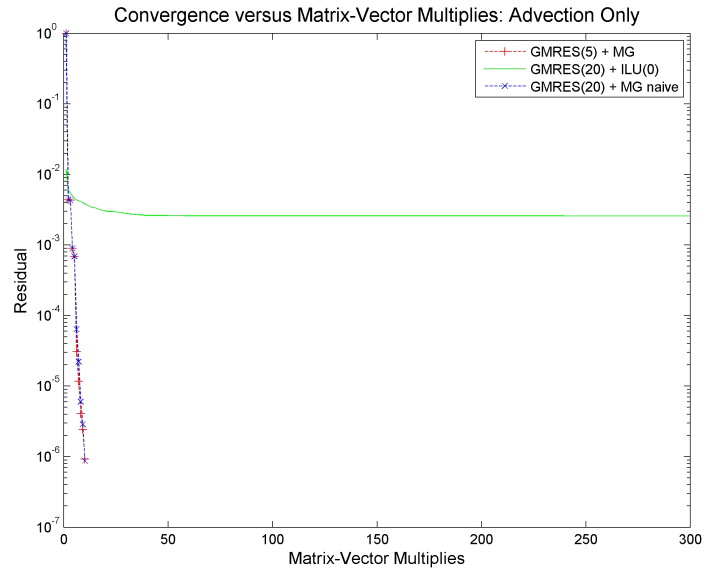

Figure C-10: Convergence history of GMRES(m) solver using different preconditioners. Here the naive MG preconditioner is used to precondition the GMRES(m) smoother for a pure advection case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

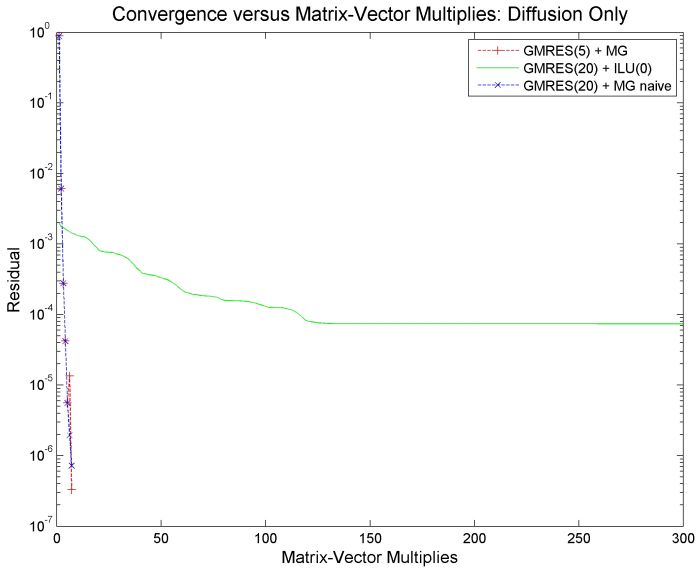

Figure C-11: Convergence history of GMRES(m) solver using different preconditioners. Here the naive MG preconditioner is used to precondition the GMRES(m) smoother for a pure diffusion case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

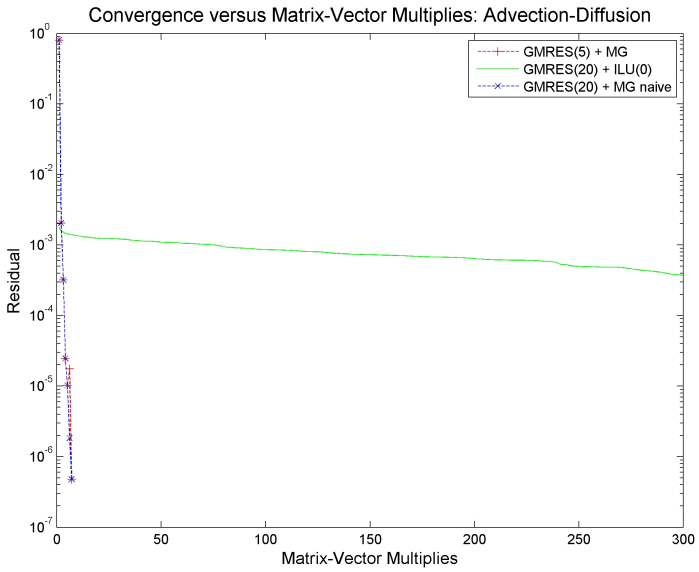

Figure C-12: Convergence history of GMRES(m) solver using different preconditioners. Here the naive MG preconditioner is used to precondition the GMRES(m) smoother for advection-diffusion case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

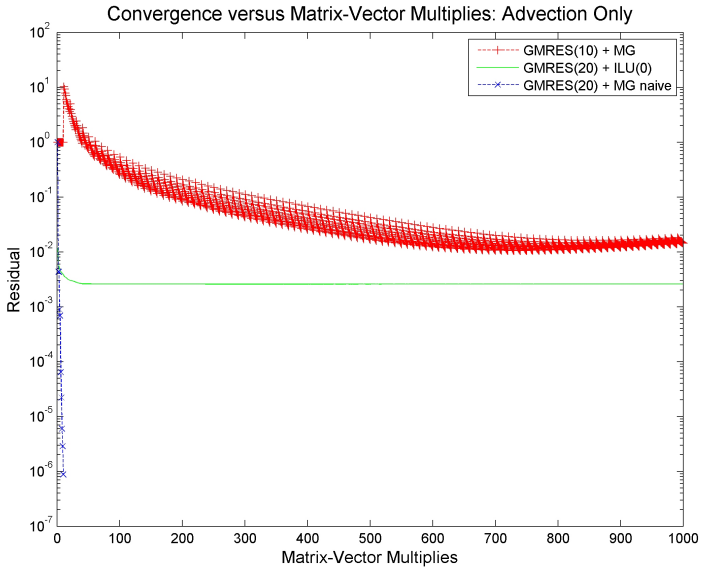

Figure C-13: Convergence history of GMRES(m) solver using different preconditioners. Here no preconditioner is used to precondition the GMRES(m) smoother for a pure advection case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

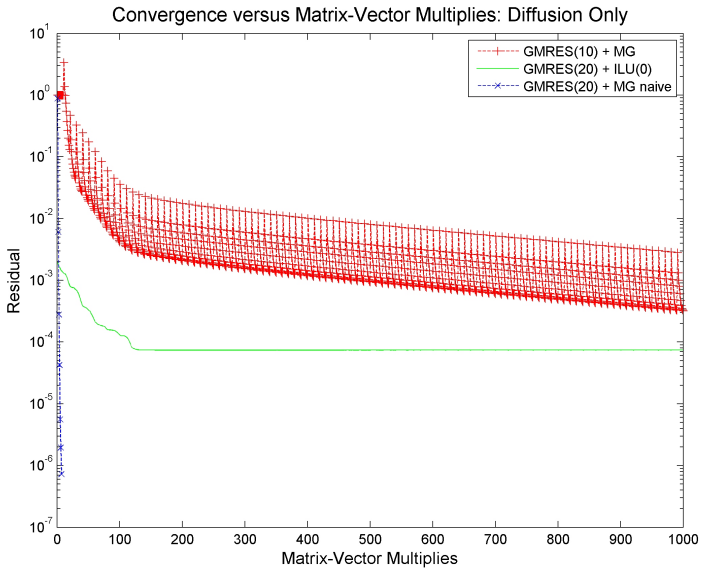

Figure C-14: Convergence history of GMRES(m) solver using different preconditioners. Here no preconditioner is used to precondition the GMRES(m) smoother for a pure diffusion case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

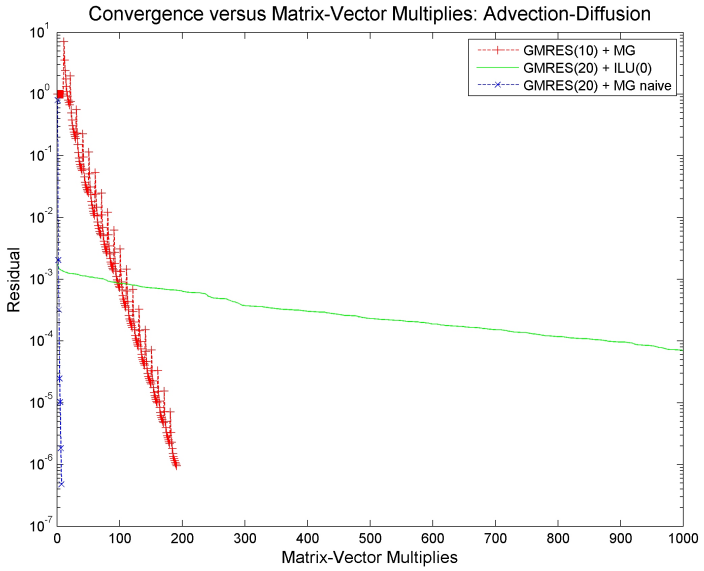

Figure C-15: Convergence history of GMRES(m) solver using different preconditioners. Here no preconditioner is used to precondition the GMRES(m) smoother for advection-diffusion case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

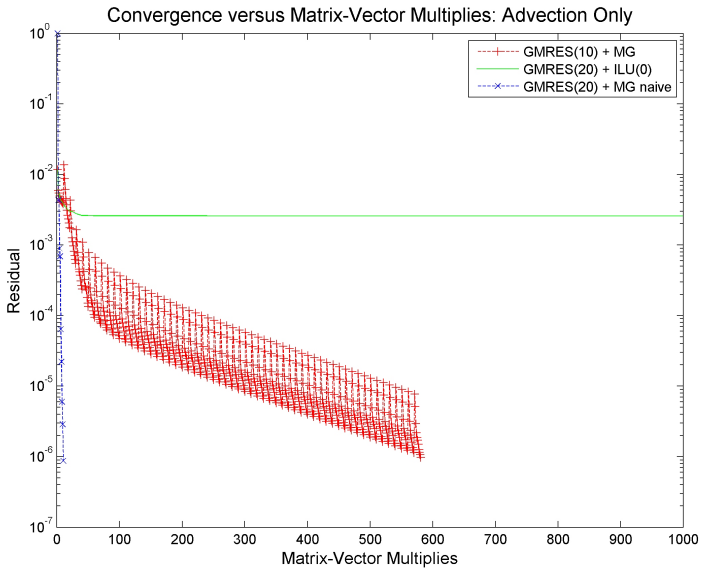

Figure C-16: Convergence history of GMRES(m) solver using different preconditioners. Here the ILU(0) preconditioner is used to precondition the GMRES(m) smoother for a pure advection case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

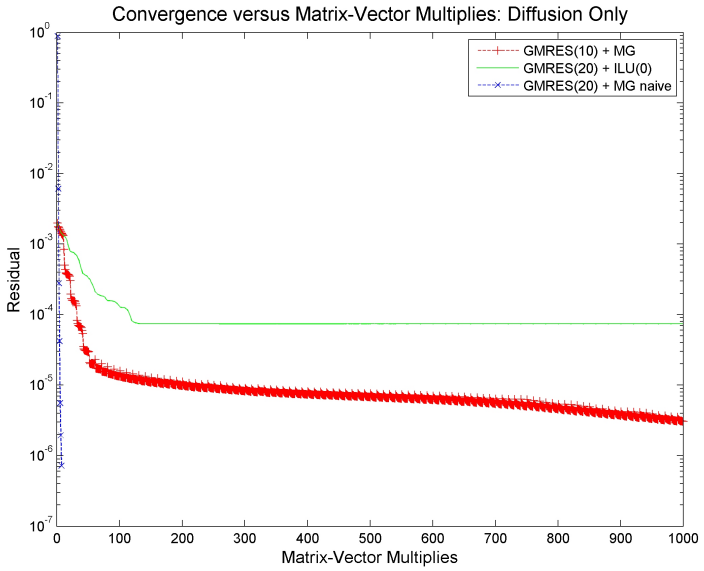

Figure C-17: Convergence history of GMRES(m) solver using different preconditioners. Here the ILU(0) preconditioner is used to precondition the GMRES(m) smoother for a pure diffusion case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

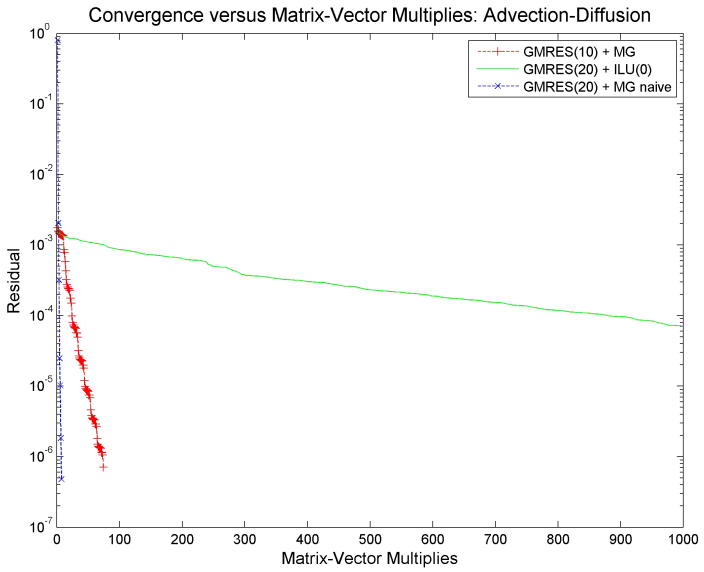

Figure C-18: Convergence history of GMRES(m) solver using different preconditioners. Here ILU(0) preconditioner is used to precondition the GMRES(m) smoother for advection-diffusion case with a proper p-MG implementation. Second order basis functions are used on the fine grid.

Bibliography

Adcroft, A., Campin, J.-M., Hill, C., and Marshall, J. (2004). Implementation of an atmosphere-ocean general circulation model on the expanded spherical cube. *Mon. Wea. Rev.*, 132 (12):2845–2863.

Agarwal, Á. (2009). Statistical field estimation and scale estimation for complex coastal regions and archipelagos. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering.

Arnold, D. N. (1982). An interior penalty finite element method with discontinuous elements. *SIAM Journal of Numerical Analysis*, 19:724–760.

Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, L. D. (2002). Unified analysis of discontinuous galerkin methods for elliptic problems. *SIAM Journal of Numerical Analysis*, 39(5):1749–1779.

Ascher, U. M., Ruuth, S. J., and Spiteri, R. J. (1997). Implicit-explicit runge-kutta methods for time-dependent partial differential equations. *Appl. Numer. Math.*, 25(2-3):151–167.

Baptista, A. M. and Zhang, Y.-L. (2008). SELFE: A semi-implicit eulerian-lagrangian finite-element model for cross-scale ocean circulation. *Ocean Modelling*, 21(3-4).

Baptista, A. M., Zhang, Y.-L., Chawla, A., Zulauf, M. A., Seaton, C., Myers, E. ., Kindle, J., Wilkin, M., Burla, M., and Turner, P. J. (2005). A cross-scale model for 3d baroclinic circulation in estuary-plume-shelf systems: Ii. application to the columbia river. *Continental Shelf Research*, 25:935–972.

Barragy, E. J. and Walters, R. A. (1998). Parallel iterative solution for h and p approximations of the shallow water equations. *Advances in Water Resources*, 21(5):327–337.

Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., R. Pozo, C. R., and der Vorst, H. V. (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA.

Bassi, F. and Rebay, S. (1997). A high-order accurate discontinuous galerkin finite element method for the numerical solution of the compressible navier-stokes equations. *Journal of Computational Physics*, 131:267–279.

Bernard, Ṕ.-E. (2008). Discontinuous galerkin methods for geophysical flow modeling. Phd thesis, Université Catholique de Louvain, Faculté des Sciences Appliqués.

Bernard, P. E., Deleersnijder, E., Legat, V., and Remacle, J. F. (2008). Dispersion analysis of discontinuous galerkin schemes applied to poincare, kelvin and rossby waves. *Journal of Scientific Computing*, 34:26–47.

Besiktepe, S., Lermusiaux, P., and Robinson, A. (2002). Coupled physical and biogeochemical data-driven simulations of massachusetts bay in late summer: real-time and postcruise data assimilation. *Journal of Marine Systems*, 40:171–212.

Besiktepe, S., Lermusiaux, P., and Robinson, A. (2003). Coupled physical and biogeochemical data driven simulations of massachusetts bay in late summer: real-time and post-cruise data assimilation. *J. of Marine Sys.*, 40:171–212.

Blain, C. A., Westerink, J. J., Luettich, R. A. J., and Scheffner, N. W. ADCIRC: An advanced three-dimensional circulation model for shelves coasts and estuaries, report 4: Hurrican storm surge modeling using large domains. Dredging Research Program Technical Report DRP-92-6, U.S. Army Engineers Waterways Experiment Station, Vicksburg, MS.

Blanton, B., Seim, H., Luettich, R., Lynch, D., Werner, F., Smith, K., Voulgaris, G., Bingham, F., and Way, F. (2004). Barotropic tides in the south atlantic bight. *Journal of Geophysical Research*, 109(C12024):17 pp.

Bleck, R. and Smith, L. (1990). A wind-driven isopycnic co-ordinate model of the north and equatorial atlantic ocean, 1: Model development and supporting experiments. *Journal of Geophysical Research*, 95:32733285.

Böning, C., Timmermann, R., Danilov, S., Schröter, J., and Boebel, O. (2006). A global finite element ocean model: Circulation and bottom pressure anomalies in the south atlantic. In *European Geosciences Union General Assembly*, Vienna, Austria.

Bryan, K. (1969). A numerical method for the study of the circulation of the world ocean. *Journal of Computational Physics*, 4:347–376.

Bunya, S., Westerink, J. J., and Yoshimura, S. (2005). Discontinuous boundary implementation for the shallow water equations. *International Journal for Numerical Methods in Fluids*, 47(12):1451–1468.

Burton, Ĺ. J. (2009). Modeling coupled physics and biology in ocean straits with application to the san bernardino strait in the philippine archipelago. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering.

Carpenter, M. and Kennedy, C. (1994). Fourth-order 2n-storage runge-kutta schemes. NASA Report TM 109112, NASA Langley Research Center.

Castillo, P., Cockburn, B., Perugia, I., and Schötzau, D. (2000). An a priori error analysis of the local discontinuous galerkin method for elliptic problems. *SIAM Journal of Numerical Analysis*, 38(5):16761706.

Casulli, V. (1999). A semi-implicit finite difference method for non-hydrostatic, free-surface flows. *International Journal for Numerical Methods in Fluids*, 30(4):425–440.

Casulli, V. and Walters, R. A. (2000). An unstructured grid, three-dimensional model based on the shallow water equations. *International Journal for Numerical Methods in Fluids*, 32(3):331–348.

Casulli, V. and Zanolli, P. (2002). Semi-implicit numerical modeling of nonhydrostatic free-surface flows for environmental problems. *Mathematical and Computer Modelling*, 36(9-10):1131–1149.

Casulli, V. and Zanolli, P. (2005). High resolution methods for multidimensional advection-diffusion problems in free-surface hydrodynamics. *Ocean Modelling*, 10(1-2):137–151.

Chapra, S. C. and Canale, R. P. (2006). *Numerical Methods for Engineers*. McGraw-Hill Higher Education, Boston, MA.

Chen, C. S., Huang, H. S., Beardsley, R. C., Liu, H. D., Xu, Q. C., and Cowles, G. (2007). A finite volume numerical approach for coastal ocean circulation studies: Comparisons with finite difference models. *Journal of Geophysical Research-Oceans*, 112(C3).

Chen, C. S., Liu, H. D., and Beardsley, R. C. (2003). An unstructured grid, finite-volume, three-dimensional, primitive equations ocean model: Application to coastal ocean and estuaries. *Journal of Atmospheric and Oceanic Technology*, 20(1):159–186.

Chen, C. S., Zhao, L. Z., Cowles, G., and Rothschild, B. (2008). Critical issues for circulation modeling of narragansett bay and mount hope bay. *Science for Ecosystem-Based Management: Narragansett Bay in the 21st Century*, pages 281–300. Desbonnet, A CostaPierce, BA Symposium on State of Science Knowledge of Nutrients in Narragansett Bay NOV, 2004 Block Isl, RI.

Cockburn, B., Gopalakrishnan, J., and Lazarov, R. (2009). Unified hybridization of discontinuous galerkin, mixed, and continuous galerkin methods for second order elliptic problems. *Siam Journal on Numerical Analysis*, 47(2):1319–1365.

Cockburn, B., Hou, S. C., and Shu, C. W. (1990). The runge-kutta local projection discontinuous galerkin finite-element method for conservation-laws .4. the multidimensional case. *Mathematics of Computation*, 54(190):545–581.

Cockburn, B., Lin, S. Y., and Shu, C. W. (1989). Tvb runge-kutta local projection discontinuous galerkin finite-element method for conservation-laws .3. one-dimensional systems. *Journal of Computational Physics*, 84(1):90–113.

Cockburn, B. and Shu, C. W. (1989). Tvb runge-kutta local projection discontinuous galerkin finite-element method for conservation-laws .2. general framework. *Mathematics of Computation*, 52(186):411–435.

Cockburn, B. and Shu, C. W. (1998a). The local discontinuous galerkin method for time-dependent convection-diffusion systems. *SIAM Journal of Numerical Analysis*, 35(6):2440–2463.

Cockburn, B. and Shu, C. W. (1998b). The runge-kutta discontinuous galerkin method for conservation laws v - multidimensional systems. *Journal of Computational Physics*, 141(2):199–224.

Danilov, S., Kivman, G., and Schroter, J. (2004). A finite-element ocean model: principles and evaluation. *Ocean Modelling*, 6(2):125–150.

Dawson, C. and Proft, J. (2002). Discontinuous and coupled continuous/discontinuous galerkin methods for the shallow water equations. *Computer Methods in Applied Mechanics and Engineering*, 191(41-42):4721–4746.

Fennel, W. and Neumann, T. (2004). *Introduction to the modelling of marine ecosystems*.

Feyen, J., Hess, K., Spargo, E., Wong, A., White, S., Sellars, J., and Gill, S. (2006). Development of a continuous bathymetric/topographi unstructured coastal flooding model to study sea level rise in north carolina. In Spaulding, M. and et al., editors, *Proc. Of the 9th Intl. Conf. on Estuarine and Coastal Modeling*. ASCE.

Fidkowski, K. J., Oliver, T. A., Lu, J., and Darmofal, D. L. (2005). p-multigrid solution of high-order discontinuous galerkin discretizations of the compressible navierstokes equations. *Journal of Computational Physics*, 207:91–113.

Flierl, G. and McGillicuddy, D. (2002). Mesoscale and submesoscale physcial-biological interactions. *The Sea*, 12:1–74.

Forbes, C., Fleming, J., Mattocks, C., Fulcher, C., Luettich, R., Westerink, J., and Bunya, S. (2007). New developments in the adcirc community model. volume 324, pages 23–23. ASCE.

Ford, R., Pain, C. C., Piggott, M. D., Goddard, A. J. H., de Oliveira, C. R. E., and Umpleby, A. P. (2004a). A nonhydrostatic finite-element model for three-dimensional stratified oceanic flows. part i: Model formulation. *Monthly Weather Review*, 132(12):2816–2831.

Ford, R., Pain, C. C., Piggott, M. D., Goddard, A. J. H., de Oliveira, C. R. E., and Umpleby, A. P. (2004b). A nonhydrostatic finite-element model for three-dimensional stratified oceanic flows. part ii: Model validation. *Monthly Weather Review*, 132(12):2832–2844.

Foreman, M. G. G., Stucchi, D. J., Zhang, Y., and Baptista, A. M. (2006). Estuarine and tidal currents in the broughton archipelago. *Atmosphere-Ocean*, 44(1):47–63.

Freeman, N., Hale, A., and Danard, M. (1972). A modified sigma equations approach to the numerical modelling of great lakes hydrodynamics. *Journal of Geophysical Research*, 77(6):10501060.

Freund, R. W. and Nachtigal, N. M. (1991). Qmr: A quasi-minimal residual method for non-hermitian linear systems. *SIAM Journal of Numerical Mathematics*, 60:315–339.

Fringer, O. B. ., McWilliams, J., and Street, R. L. . (2006a). A new hybrid model for coastal simulations. *Oceanography*, 19(1):64–77.

Fringer, O. B., Gerritsen, M., and Street, R. L. (2006b). An unstructured-grid, finite-volume, nonhydrostatic, parallel coastal-ocean simulator. *Ocean Modelling*, 14(3-4):139–278.

Geuzaine, C. and Remacle, J.-F. (2009). Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, (Accepted):Published online.

Gica, E., Teng, M. H., Luettich, R. A., Cheung, K. F., Blain, C. A., Wu, C.-S., and Scheffner, N. W. (2001). Numerical modeling of strom surge generated by hurrican iniki in hawaii. In WAVES *2001, proceedings of the 4th International Symposium on Ocean Wave Measurement and Analysis*, number 2, pages 1555–1564, San Francisco, California. ASCE.

Gottlieb, S., Shu, C. W., and Tadmor, E. (2001). Strong stability preserving high order time discretization methods. *SIAM*, (43):89–112.

Griffies, S. M., Böning, C., Bryan, F. O., Chassignet, E. P., Gerdes, R., Hasumi, H., Hirst, A., Treguier, A.-M., and Webb, D. (2000). Developments in ocean climate modelling. *Ocean Modelling*, 2:123–192.

Ham, D. A. (2006). *On techniques for modelling coastal and ocean flow with unstructured meshes*. PhD dissertation, TU Delft.

Heaney, K., Gawarkiewicz, G., Duda, T., and Lermusiaux, P. (2007). Non-linear optimization of autonomous undersea vehicle sampling strategies for oceanographic data assimilation. special issue on underwater robotics. *Journal of Field Robotics*, 24(6):437–448.

Hesthaven, J. S. and Warburton, T. (2002). Nodal high-order methods on unstructured grids. *Journal of Computational Physics*, 181:186–221.

Hesthaven, J. S. and Warburton, T. (2008). *Nodal Discontinuous Galerkin Methods*, volume 54 of *Texts in Applied Mathematics*. Springer, New York, NY.

Iskandarani, M., Haidvogel, D. B., and Boyd, J. P. (1995). A staggered spectral element model for the shallow water equations. *International Journal for Numerical Methods in Fluids*, 20(5):393–414.

Iskandarani, M., Haidvogel, D. B., and Levin, J. C. (2003). A three-dimensional spectral element model for the solution of the hydrostatic primitive equations. *Journal of Computational Physics*, 186:397425.

Jachec, S. M., Fringer, O. B., Gerritsen, M. G., and Street, R. L. (2006). Numerical simulation of internal tides and the resulting energetics within monterey bay and the surrounding area. *Geophysical Research Letters*, 33(12).

Jachec, S. M., Fringer, O. B., Street, R. L., and Gerritsen, M. G. (2007). Effects of grid resolution on the simulation of internal tides. *International Journal of Offshore and Polar Engineering*, 17(2):105–111.

Jarosz, E., Blain, C. A., Murray, S. P., and Inoue, M. (2005). Bartotropic tides in the bab el mandab strait - numerical simulations. *Continental Shelf Research*, 25(10):1225–1247.

Karniadakis, G. E. and Sherwin, S. (2005). *Spectral/hp Element Methods for Computational Fluid Dynamics*. Numerical Mathematics and Scientific Computation. Oxford Science Publications, New York, NY, second edition.

Kennedy, C. A. and Carpenter, M. H. (2003). Additive runge-kutta schemes for convection-diffusion-reaction equations. *Applied Numerical Mathematics*, 44(1-2):139–181.

Koornwinder, T. H. (1991). Askey-wilson polynomials for root systems of type bc, in hypergeo-metric functions on domains of positivity, jack polynomials, and applications. *Contemp. Math., Amer. Math. Soc.*, 138:189–204.

Kubatko, E. J., Bunya, S., Dawson, C., Westerink, J. J., and Mirabito, C. (2009). A performance comparison of continuous and discontinuous finite element shallow water models. *Journal of Scientific Computing*, 40:315–339.

Kubatko, E. J., Westerink, J. J., and Dawson, C. (2006). An unstructured grid morphodynamic model with a discontinuous galerkin method for bed evolution. *Ocean Modelling*, 15(1-2):71–89.

Lane, E. M. and Walters, R. A. (2009). Verification of ricom for storm surge forecasting. *Marine Geodesy*, 32:118132.

Le Roux, D. Y. (2005). Dispersion relation analysis of the p-1(nc)-p-1 finite-element pair in shallow-water models. *Siam Journal on Scientific Computing*, 27(2):394–414.

Le Roux, D. Y., Lin, C. A., and Staniforth, A. (2000). A semi-implicit semi-lagrangian finite-element shallow-water ocean model. *Monthly Weather Review*, 128(5):1384–1401.

Legrand, S., Deleersnijder, E., Delhez, E., and Legat, V. (2007). Unstructured, anisotropic mesh generation for the northwestern european continental shelf, the continental slope and the neighbouring ocean. *Continental Shelf Research*, 27(9):1344–1356.

Lemusiaux, P. F. J. (1999). Data assimilation via error subspace statistical estimation. part ii: Middle atlantic bight shelfbreak front simulations and esse validation. *Monthly Weather Review*, 127(7):1408–1432.

Lemusiaux, P. F. J. (2007). Adaptive modeling, adaptive data assimilation and adaptive sampling. *Eds. Physica D*, 230:172–196.

LeVeque, R. (2002). *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, UK.

Leveque, R. J. (1996). High-resolution conservative algorithms for advection in incompressible flow. *Siam Journal on Numerical Analysis*, 33(2):627–665.

Liu, W. C., Chen, W. B., Cheng, R. T., and Hsu, M. H. (2008). Modelling the impact of wind stress and river discharge on danshuei river plume. *Applied Mathematical Modelling*, 32(7):1255–1280. Liu, Wen-Cheng Chen, Wei-Bo Cheng, Ralph T. Hsu, Ming-Hsi 36 ELSEVIER SCIENCE INC 298NO.

Logutov, O. G. and Lemusiaux, P. F. J. (2008). Inverse barotropic tidal estimation for regional ocean applications. *Ocean Modelling*, 25:17–34.

Luettich, R. A. J., Hench, J. L., Fulcher, C. W., Werner, F. E., Blanton, B. O., and Churchill, J. H. (1999). Barotropic tidal and wind driven larval transport in the vicinity of a barrier island inlet. *Fisheries Oceanography*, 8(Suppl. 2):190–209.

Luettich, R. A. J. and Westerink, J. J. (1995). Implementation and testing of elemental flooding and drying in the ADCIRC hydrodynamic model. Final report, 8/95, contract # DACW39 − 94 − M − 5869.

Marshall, J., Adcroft, A., Hill, C., Perelman, L., and Heisey, C. (1997a). A finite-volume, incompressible navier stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research-Oceans*, 102(C3):5753–5766.

Marshall, J., Hill, C., Perelman, L., and Adcroft, A. (1997b). Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling. *Journal of Geophysical Research-Oceans*, 102(C3):5733–5752.

Marshall, J., Jones, H., and Hill, C. (1998). Efficient ocean modeling using non-hydrostatic algorithms. *Journal of Marine Systems*, 18(1-3):115–134.

Molcard, A., Pinardi, N., Iskandarani, M., and Haidvogel, D. B. (2002). Wind driven circulation of the mediterranean sea simulated with a spectral element ocean model. *Dynamics of Atmospheres and Oceans*, 35:97130.

Myers, E. and Aikman, F. (2003). A forecast circulation model of the st. johns river, florida. In *8th Estuarine and Coastal Modeling Conference*, Monterey Bay, CA, USA.

Nerger, L., Danilov, S., Hiller, W., and Schröter, J. (2006). Using sea-level data to constrain a finite-element primitive-equation ocean model with a local seik filter. *Ocean Dynamics*, 56(5-6):634–649.

Nguyen, N. C., Peraire, J., and Cockburn, B. (2009). An implicit high-order hybridizable discontinuous galerkin method for linear convection-diffusion equations. *Journal of Computational Physics*, 228(9):3232–3254.

Pain, C. C., Piggott, M. D., Goddard, A. J. H., Fang, F., Gorman, G. J., Marshall, D. P., Eaton, M. D., Power, P. W., and de Oliveira, C. R. E. (2005). Three-dimensional unstructured mesh ocean modelling. *Ocean Modelling*, 10(1-2):5–33.

Peraire, J. and Persson, P. O. (2007). The compact discontinuous galerkin (cdg) method for elliptic problems. *Siam Journal on Scientific Computing*, 30(4):1806–1824.

Persson, P.-O. and Peraire, J. (2006). An efficient low memory implicit dg algorithm for time dependent problems. In *44th AIAA Aerospace Sciences Meeting and Exhibit*, number AIAA-2006-0113, Reno, Nevada.

Persson, P.-O. and Peraire, J. (2008). Newton-gmres preconditioning for discontinuous galerkin discretizations of the navier-stokes equations. *SIAM Journal of Scientific Computing*, 30(6):2709–2733.

Pietrzak, J., Jakobson, J. B., Burchard, H., Vested, H. J., and Petersen, O. (2002). A three-dimensional hydrostatic model for coastal and ocean modelling using a generalised topography following co-ordinate system. *Ocean Modelling*, 4:173205.

Piggott, M. D., Gorman, G. J., Pain, C. C., Allison, P. A., Candy, A. S., Martin, B. T., and Wells, M. R. (2008). A new computational framework for multi-scale ocean modelling based on adapting unstructured meshes. *International Journal for Numerical Methods in Fluids*, 56(8):1003–1015.

Piggott, M. D., Pain, C. C., Gorman, G. J., Power, P. W., and Goddard, A. J. H. (2005). h, r, and hr adaptivity with applications in numerical ocean modelling. *Ocean Modelling*, 10(1-2):95–113.

Pinto, L. L., Oliveira, A., Fortunato, A. B., and Baptista, A. M. (2003). Analysis of the stratification in the guadiana estuary. In *8th Estuarine and Coastal Modeling Conference*, Monterey Bay, CA, USA.

Power, P. W., Pain, C. C., Piggott, M. D., Fang, F., Gorman, G. J., Umpleby, A. P., Goddard, A. J. H., and Navon, I. M. (2006). Adjoint a posteriori error measures for anisotropic mesh optimisation. *Computers & Mathematics with Applications*, 52(8-9):1213–1242.

Reed, W. H. and Hill, T. R. (1973). Triangular mesh methods for the neutron transport equations. Los Alamos Scientific Laboratory Report LA-UR-73-479, Fanstord University.

Remacle, J. F., Frazao, S. S., Li, X. G., and Shephard, M. S. (2006). An adaptive discretization of shallow-water equations based on discontinuous galerkin methods. *International Journal for Numerical Methods in Fluids*, 52(8):903–923.

Remacle, J. F., Li, X. R., Shephard, M. S., and Flaherty, J. E. (2005). Anisotropic adaptive simulation of transient flows using discontinuous galerkin methods. *International Journal for Numerical Methods in Engineering*, 62(7):899–923.

Robinson, A. and Lermusiaux, P. (2003). Prediction systems with data assimilation for coupled ocean science and ocean acoustics. *Theoretical and Computational Acoustics*, pages 325–342.

Robinson, C. L. K., Morrison, J., and Foreman, M. G. G. (2005). Oceanographic connectivity among marine protected areas on the north coast of british columbia, canada. *Canadian Journal of Fisheries and Aquatic Sciences*, 62(6):1350–1362.

Shen, J., Wang, H., Sisson, M., and Gong, W. (2006a). Storm tide simulation in the chesapeake bay using an unstructured grid model. *Estuarine Coastal and Shelf Science*, 68(1-2):1–16. 43 ACADEMIC PRESS LTD ELSEVIER SCIENCE LTD 057OG.

Shen, J., Zhang, K. Q., Xiao, C. Y., and Gong, W. P. (2006b). Improved prediction of storm surge inundation with a high-resolution unstructured grid model. *Journal of Coastal Research*, 22(6):1309–1319. Shen, Jian Zhang, Keqi Xiao, Chengyou Gong, Wenping 26 COASTAL EDUCATION & RESEARCH FOUNDATION 114WA.

Sleijpen, G. L. G. (2009). Gerhard sleijpen: Department of mathematics, universiteit utrecht. *http://www.math.uu.nl/people/sleijpen/*.

Sleijpen, G. L. G. and Fokkema, D. R. (1993). Bicgstab(l) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions On Numerical Analysis*, 1:11–32.

Slingo, J., Bates, K., Nikiforakis, N., Piggott, M., Roberts, M., Shaffrey, L., Stevens, I., Vidale, P. L., and Weller, H. (2009). Developing the next-generation climate

system models: challenges and achievements. *Philosophical Transactions of the Royal Society a-Mathematical Physical and Engineering Sciences*, 367(1890):815–831. Discussion Meeting on the Environmental eScience Revolution APR 07-08, 2008 Royal Soc, London, ENGLAND.

Spall, M. and Robinson, A. (1990). Regional primitive equation studies of the gulf stream meander and ring formation region. *Journal of Physical Oceanography*, 20:9851016.

Trefethen, L. N. and Bau, D. (1997). *Numerical Linear Algebra*. SIAM, Philadelphia, PA.

Venayagamoorthy, S. K. and Fringer, O. B. (2005). Nonhydrostatic and nonlinear contributions to the energy flux budget in nonlinear internal waves. *Geophysical Research Letters*, 32(15).

Walters, R. A. (2005a). Coastal ocean models: two useful finite element methods. *Continental Shelf Research*, 25(7-8):775–793.

Walters, R. A. (2005b). A semi-implicit finite element model for non-hydrostatic (dispersive) surface waves. *International Journal for Numerical Methods in Fluids*, 49(7):721–737.

Walters, R. A. (2006). Design considerations for a finite element coastal ocean model. *Ocean Modelling*, 15(1-2):90–100.

Walters, R. A. and Barragy, E. J. (1997). Comparison of h and p finite element approximations of the shallow water equations. *International Journal for Numerical Methods in Fluids*, 24(1):61–79.

Walters, R. A., Lane, E. M., and Henry, R. F. (2007). Semi-lagrangian methods for a finite element coastal ocean model. *Ocean Modelling*, 19:112–124.

Wang, Q., Danilov, S., and Schröter, J. (2008). Finite element ocean circulation model based on triangular prismatic elements, with application in studying the effect of topography representation. *Journal of Geophysical Research-Oceans*, 113(C5):21. Wang, Q. Danilov, S. Schroeter, J. 92 AMER GEOPHYSICAL UNION 300NA.

Wang, Q., Danilov, S., and Schröter, J. (2009). Bottom water formation in the southern weddell sea and the influence of submarine ridges: Idealized numerical simulations. *Ocean Modelling*, 28(1-3):50–59.

Web-ADCIRC (2006). ADCIRC development group. *http://www.nd.edu/ adcirc/index.htm*.

Web-AWI (2009). FEOM website. *http://www.awi.de/ en/ research/research_divisions/ climate_science/ ocean_dynamics/ community_ocean_model_com/ finite_element_ocean_model_feom/*.

Web-CORRIE (2009). CORRIE group website. *http://www.ccalmr.ogi.edu/CORIE/*.

Web-DELFT (2009). DELFIN/FINEL group website. *http:// www.tudelft.nl/ live/ pagina.jsp?id=dcfaf183-e1c7-47b5-abba-b1650947e2c8&lang=en*.

Web-FEOM (2009). COM development website. *http://aforge.awi.de/gf/project/com/*.

Web-FVCOM (2009). FVCOM development website. *http://fvcom.smast.umassd.edu/ FVCOM/*.

Web-ICOM (2008). ICOM development website. *http://amcg.ese.ic.ac.uk/ index.php?title=ICOM*.

Web-MSEAS (2009). Multidisciplinary simulation, estimation, and assimilation systems. *http://mseas.mit.edu/*.

Web-SEOM (2009). SEOM development website. *http://ostrogoth.rsmas.miami.edu/ ~mohamed/ SEOM/ seom_index.html*.

Web-SLIM (2009). SLIM development website. *http://sites.uclouvain.be/slim/*.

Web-SUNTANS (2009). SUNTANS development website. *http://suntans.stanford.edu/*.

Web-UNTRIM (2009). UnTRIM website. *http://www.baw.de/ vip/ en/ departments/ department_k/ methods/ hnm/ untrim/ hnm_untrim-en.html*.

Westerink, J. J., Jr., R. A. L., Feyen, J. C., Atkinson, J. H., Dawson, C., Powell, M. D., Dunion, J. P., Roberts, H. J., Kubatko, E. J., and Pourtaheri, H. (2007). A basin to channel scale unstructured grid hurrican storm surge model as implemented for southern louisiana. *Monthly Weather Review*, 136:833–864.

Westerink, J. J., Luettich, R. A. J., and Muccino, J. (1994). Modeling tides in the wester north atlantic using unstructured graded grids. *Tellus*, 46(A):178–199.

White, L. (2008). Tracer conservation for three-dimensional, finite-element, free-surface, ocean modeling on moving prismatic meshes. *Monthly Weather Review*, 136(2):420–442. White, Laurent.

White, L., Deleersnijder, E., and Legat, V. (2008). A three-dimensional unstructured mesh finite element shallow-water model, with application to the flows around an island and in a wind-driven, elongated basin. *Ocean Modelling*, 22(1-2):26–47.

Wunsch, C., Haidvogel, D. B., Iskandarani, M., and Hughes, R. (1997). Dynamics of the long-period tides. *Progress in Oceanography*, 40:80108.

Xue, P. F., Chen, C. S., Ding, P. X., Beardsley, R. C., Lin, H. C., Ge, J. Z., and Kong, Y. Z. (2009). Saltwater intrusion into the changjiang river: A model-guided mechanism study. *Journal of Geophysical Research-Oceans*, 114.

Yan, J. and Shu, C.-W. (2002). Local discontinuous galerkin methods for partial differential equations with higher order derivatives. *Journal of Scientific Computing*, 17:27–47.

Zhang, Y.-L., Baptista, A. M., and Myers, E. P. (2004). A cross-scale model for 3d baroclinic circulation in estuary-plume-shelf systems: I. formulation and skill assessment. *Continental Shelf Research*, 24:2187–2214.

Zhao, L. Z., Chen, C. S., and Cowles, G. (2006). Tidal flushing and eddy shedding in mount hope bay and narragansett bay: An application of fvcom. *Journal of Geophysical Research-Oceans*, 111(C10).