Efficient solvers for the implicit time integration of matrix-free high-order methods

by

Michael Franco

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Applied Mathematics

and the Designated Emphasis

in

Computational and Data Science and Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Per-Olof Persson, Chair
Professor Jon Wilkening
Professor Katherine Yelick

Summer 2022

Efficient solvers for the implicit time integration of matrix-free high-order methods

Abstract

Efficient solvers for the implicit time integration of matrix-free high-order methods

by

Michael Franco

Doctor of Philosophy in Applied Mathematics

and the Designated Emphasis in

Computational and Data Science and Engineering

University of California, Berkeley

Professor Per-Olof Persson, Chair

In this thesis, we develop and study efficient solvers for high-order Galerkin methods applied to fluid flow problems. Many flow problems necessitate implicit time-integration schemes in order to be practical. Implicit-in-time discretizations require the solution of nonlinear algebraic systems each time step, which are often in-turn solved by linear solvers. Therefore, the performance of implicit-in-time solvers is largely determined by the performance of the underlying linear solvers.

One approach to create efficient methods is to work with matrix-free operators. Because assembling the underlying discretization matrix can be prohibitively expensive in terms of computational complexity and memory, matrix-free operators are an attractive alternative. These operators replace the matrix-vector products with on-the-fly sum-factorization evaluations of the discretized differential operators instead. Indeed, their high arithmetic intensity makes these operators particularly well suited for modern graphics processing units (GPU) and GPU-accelerated architectures.

These matrix-free operators are particularly challenging to precondition, however, because they by design do not allow access to the underlying matrix entries. We create a suite of efficient matrix-free preconditioners for a range of fluid flow problems that are robust with respect to polynomial degree and mesh size. The main building block solver extends sparse, low-order refined preconditioners with parallel subspace corrections. This work tackles Poisson problems, saddle-point Stokes systems, and the incompressible Navier-Stokes equations in two and three spatial dimensions.

A different set of problems exhibit *geometrically localized stiffness*, where convergence rates are degraded in a localized subregion of the mesh. Generic preconditioners do not perform well across the entire domain because of mesh size, mesh anisotropy, highly variable coefficients, or more challenging physics in the subregion. Therefore, we seek to save costs by utilizing cheap preconditioners for most of the mesh and only focus our effort on the less expensive subregion problem. Our iterative subregion correction preconditioners correct naive preconditioners with an adaptive inner subregion iteration to reduce the number of costly global iterations. This work demonstrates performance on basic convection-diffusion problems, high Reynolds number compressible flow problems, and a 30° angle of attack problem with massively separated flow.

To Benoit “Benoit B. Mandelbrot” Mandelbrot

Contents

List of Figures

List of Tables

Acknowledgments

This dissertation thesis would not have been possible without the help of others, and so I would like to thank all those who supported me along the way.

First of all, thank you to my advisor Per-Olof Persson for your guidance over these past years. Your deep understanding and inquisitive personality paved the way to many of the breakthroughs in this work. I have better learned how to ask the correct questions while discussing the state of the field with you. Thank you for shaping my interests in fluid dynamics and numerical solvers as I have developed as a researcher.

A huge thanks is owed to Will Pazner for sharing so much of your knowledge and time with me over the years. You have been a source of inspiration to me, both in your technical skills as well as your collaborative generosity. Thank you for challenging me to develop a deeper understanding of the tools and algorithms I used throughout this Ph.D.

Next I would like to thank my fellow Berkeley students for ensuring this journey was shared together. Thanks to Andrew Shi and Nick Bhattacharya for our research and not-so-research discussions. Thanks to my office buddies Yanhe Huang and Luke Corcos for being a constant source of delight, even when times were tough. Thanks to Colin Wahl for helping me keep perspective. Thanks to Yimin Lin and Kaiwen He for being such good mentees that you challenged me to improve myself. Thanks to Junior Meija, Theo Coyne, Ashritha Eswaran, and all the other members of the Berkeley Chess Club for helping further my chess journey and giving me a space to unwind.

Thanks to Jon Wilkening for directing my interests as I started my Berkeley journey. I am grateful to Dwight Raulston, John Mellor-Crummey, Tim Warburton, and Mike O'brien for leading me down my path to pursue a Ph.D. and helping guide my interest in numerical methods. Thanks to Justin Dong for allowing me to bounce mathematical ideas off you, even when across the country. Thanks to the online chess community, especially members of the Morphy Chess Club, for helping provide some sanity during this insane pandemic.

Thanks to my family for sparking a lifelong love of learning. I appreciate just how often you would listen to my explanations of my research, even when the nouns and half the verbs were foreign to you. Thanks for keeping me grounded over the years.

Finally, my biggest thanks of course goes to my wife Fran Iyer for being a constant companion throughout this journey. Without your support and love, I would never have started this Ph.D., let alone finish it. Thanks for being there with me throughout all the highs and lows. I look forward to our next adventures.

Chapter 1

Introduction

Computational fluid dynamics (CFD) is a branch of fluid mechanics that models and analyzes fluid flows on a computer. CFD is applied to a wide range of research and engineering problems including aerodynamics and aerospace analysis [66, 5], hypersonics [27], weather simulation [82, 57], engine and combustion analysis [84, 87], hemodynamics [79], and visual effects for film and video games [124]. CFD problems are mathematically based upon the Navier-Stokes equations or simplifications of these equations. In particular, turbulent flows are intrinsically unsteady, span multiple scales, and may be anisotropic [129, 16], requiring high resolution to accurately model. Yet direct numerical simulation of the Navier-Stokes equations is typically prohibitively expensive due to the Kolmogorov scale [123], and traditional approaches like Reynolds-averaged Navier-Stokes do not accurately capture certain flow transitions [15]. Therefore, large eddy simulation (LES) has proven to be a computationally tractable alternative to offer significantly higher accuracy for certain flows [16, 113].

The spatial discretization of the underlying equations governs the necessary degrees of freedom for an accurate solution. Thus, high-order finite element methods have been developed with the potential to attain higher accuracy per degree of freedom than low-order alternatives [45, 132]. Due to these potential computational savings, many high-order methods have been developed to solve a diverse range of CFD problems [37, 9, 99]. Moreover, the high arithmetic intensity of these algorithms makes them a prime target for use on graphics processing units (GPUs) and GPU-accelerated architectures of current and future supercomputers [74, 131, 25].

The discontinuous Galerkin (DG) method is a particular high-order finite element method originally introduced in 1973 by Reed and Hill to solve the neutron transport equation [108]. In the early 1990s, Cockburn, Shu, and others extended the DG method to general systems of hyperbolic conservation laws with a method of lines approach [36, 121, 40, 37, 35]. Using the DG method as the spatial discretization and explicit Runge-Kutta (RK) schemes as the temporal discretization allowed time-dependent problems involving discontinuities to be solved efficiently. Extensions to elliptic and parabolic systems were developed around the turn of the century [9, 10, 20, 38, 4]. The main benefits of DG are inherited from its generalization of both finite volume and finite element methods. In particular, it is high-order accurate, highly parallelizable, well suited to complicated geometries, and can accurately model physically relevant discontinuities [36]. Since its development, DG has been applied to a wide range of fields, including CFD [92, 99], atmospheric modeling [90], magnetohydrodynamics [134], elastic wave propagation [26, 6], and more.

The RKDG methods developed in [121, 40, 37, 35] were designed to use explicit RK time integration schemes. Explicit integration allows for cheap and highly parallelizable time steps, which only require residual evaluations [59]. However, a major drawback of explicit methods is that they are limited by the Courant–Friedrichs–Lewy (CFL) condition [41, 80]. For many CFD problems, this condition on the allowable time step is too restrictive, necessitating implicit time integration schemes for DG discretizations be used instead [7, 8]. Implicit-in-time discretizations require the solution of nonlinear algebraic systems each time step, which are often in-turn solved by linear iterative solvers. Therefore, the performance of implicit-in-time fluid flow solvers is largely determined by the performance of the underlying linear solvers.

Unfortunately, the widespread of adoption of high-order implicit-in-time methods is currently limited by their relatively expensive computational cost [132, 133]. Therefore, work must be done to speed up the fundamental building block of the methods, the underlying linear solver. Targeted performance improvements for iterative methods for linear systems typically fall under two categories: those that minimize the cost of a single iteration, and those that minimize the total number of iterations. Recent progress in the former has been accomplished by implementing faster matrix-vector kernels, replacing matrix-vector products with on-the-fly evaluations of the discretized differential operators in a matrix-free manner, and utilizing massive throughput capabilities of GPUs [74, 125, 78]. Progress in the latter has been accomplished by better preconditioning of these linear systems [102, 53, 105]. The topic of this thesis is the development and analysis of efficient solvers and preconditioners designed for implicit time integration of high-order methods. Indeed, our work on GPU acceleration and matrix-free operators falls under the first category, and work on robust matrix-free preconditioners and iterative subregion correction preconditioners falls under the second.

The structure of this thesis is as follows. Chapter 2 will formulate the DG method for systems of second-order parabolic equations. Implicit RK schemes will be introduced to solve the semi-discrete equations, ultimately leading to a fundamental linear system for which performance is critical. Chapter 3 will describe matrix-free high-order methods for incompressible fluid flow. Particular emphasis will be on the GPU implementation and performance. Next, we will detail our matrix-free preconditioning techniques for incompressible flow in Chapter 4. Chapter 5 will introduce the concept of geometrically localized stiffness and develop a class of efficient preconditioners to handle flow problems with this property. Finally, we apply these techniques to challenging large eddy simulations of turbulent flow in Chapter 6 in order to illustrate the main properties of our methods.

Chapter 2

Governing equations, discretizations, and other preliminaries

2.1 Governing equations

Throughout this thesis, we will solve several sets of governing equations. For reference, we list these partial differential equations (PDE) below. Note that they all are defined for a domain $\Omega \subset \mathbb{R}^d$ in dimension d. The PDE is well-defined once coupled with appropriate boundary conditions on $\partial\Omega$.

2.1.1 System of conservation laws

There is a general PDE of which many of the following equations are examples. A general system of conservation laws takes the form

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot F_c(\boldsymbol{u}) - \nabla \cdot F_d\left(\boldsymbol{u}, \nabla \boldsymbol{u}\right) = 0, \tag{2.1}$$

where $\boldsymbol{u} : \Omega \to \mathbb{R}^{n_c}$ is a vector of unknowns with n_c components, $F_c : \Omega \to \mathbb{R}^{n_c \times d}$ are the convective fluxes, $F_d : \Omega \to \mathbb{R}^{n_c \times d}$ are the diffusive fluxes. If $F_d = 0$, then (2.1) simplifies to a general system of hyperbolic conservation flaws, a first-order PDE.

2.1.2 Compressible Navier-Stokes

A broad spectrum of fluid dynamics falls under the mathematical umbrella of the unsteady compressible Navier-Stokes equations,

$$\begin{aligned}\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) &= 0 \\ \frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot \left(\rho \boldsymbol{u} \otimes \boldsymbol{u}^T + p \boldsymbol{I}_d - \boldsymbol{\tau}\right) &= 0 \\ \frac{\partial (\rho E)}{\partial t} + \nabla \cdot ((\rho E + p)\boldsymbol{u} + \boldsymbol{q} - \boldsymbol{\tau} \boldsymbol{u}) &= 0,\end{aligned} \tag{2.2}$$

where ρ is the fluid density, $\boldsymbol{u} : \Omega \to \mathbb{R}^d$ is the fluid velocity for dimension $d = 2, 3$, E is the total energy, p is the pressure, and $\boldsymbol{I}_d$ is the $d \times d$ identity tensor. The viscous stress tensor $\boldsymbol{\tau}$ and heat flux $\boldsymbol{q}$ are given by

$$\boldsymbol{\tau} := \mu \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T - \frac{2}{3} (\nabla \cdot \boldsymbol{u}) \boldsymbol{I}_d \right), \qquad \boldsymbol{q} := -\frac{\mu}{Pr} \nabla \left(E + \frac{p}{\rho} - \frac{1}{2} \|\boldsymbol{u}\|_2^2 \right), \tag{2.3}$$

respectively, where μ is the viscosity coefficient and $Pr = 0.72$ is the Prandtl number, which we assume to be constant. For an ideal gas, the pressure satisfies

$$p = (\gamma - 1) \rho \left(E - \frac{1}{2} \|\boldsymbol{u}\|_2^2 \right), \tag{2.4}$$

where $\gamma = 1.4$ is the adiabatic gas constant.

We can explicitly rewrite (2.2) in the form of (2.1) with the definitions

$$u := \begin{pmatrix} \rho \\ \rho \boldsymbol{u} \\ \rho E \end{pmatrix}, \quad F_c := \begin{pmatrix} \rho \boldsymbol{u} \\ \rho \boldsymbol{u} \otimes \boldsymbol{u}^T + p \boldsymbol{I}_d \\ \rho H \boldsymbol{u} \end{pmatrix}, \quad F_d := \begin{pmatrix} 0 \\ \boldsymbol{\tau} \\ \boldsymbol{\tau} \boldsymbol{u} - \boldsymbol{q} \end{pmatrix}, \tag{2.5}$$

where $H = E + p/\rho$ is the stagnation enthalpy. Thus u is a vector with $d + 2$ unknowns.

The Reynolds number is defined as

$$Re = \frac{\rho u L}{\mu}, \tag{2.6}$$

where u is the flow speed and L is the characteristic linear dimension. L is a dimensionless number determining the characteristic length of a given fluid, and so is 1 for a nondimensionalized problem. Low Reynolds number flows are dominated by laminar effects, while high Reynolds number flows transition to turbulence since smaller and smaller scales of the flow are visible.

The Mach number is defined as

$$M = \frac{u}{c}, \tag{2.7}$$

where c is the speed of sound, or 340.3 m/s under normal conditions. M is a dimensionless number determining the speed of a flow relative to the local speed of sound. The freestream mach number M_∞ defines the relative speed of the problem far away from any fixed body, such as an airfoil.

In some cases, we modify (2.2) by introducing an isentropic assumption. Since the entropy remains constant, the flow is adiabatic and reversible. For a perfect gas, the entropy satisfies

$$s = p/\rho^\gamma, \tag{2.8}$$

as described in [70]. Favorably, this equation allows us to relate pressure and density, thereby making the energy equation in (2.2) redundant. This assumption reduces the number of components of the PDE from $d+2$ to $d+1$. This simplification is an artificial compressibility model for incompressible flows and allows us to solve nearly incompressible flows without the specialized solvers required by the incompressible Navier-Stokes equations that we will discuss next.

2.1.3 Incompressible Navier-Stokes

We also consider the unsteady incompressible Navier-Stokes equations in d spatial dimensions. By assuming uniform density and viscosity across the domain, valid in the limit as $M \to 0$, (2.2) simplifies to

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} - \left(\frac{\mu}{\rho}\right) \Delta \boldsymbol{u} + \nabla p &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned} \tag{2.9}$$

The right-hand side $\boldsymbol{f}$ is a known forcing term that may or may not be 0 across the domain. We define the kinematic viscosity $\nu = \mu/\rho$. Now we can explicitly rewrite the first equation in the form of (2.1), yielding

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \left(\boldsymbol{u} \otimes \boldsymbol{u}^T + p\boldsymbol{I}_d - \nu \nabla \boldsymbol{u}\right) &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned} \tag{2.10}$$

This formulation requires the solution of a system of conservation laws with $d+1$ unknowns, coupled with the incompressibility constraint.

We also consider the steady version of (2.10),

$$\begin{aligned} \nabla \cdot \left(\boldsymbol{u} \otimes \boldsymbol{u}^T + p\boldsymbol{I}_d - \nu \nabla \boldsymbol{u}\right) &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned} \tag{2.11}$$

2.1.4 Stokes

In the low Reynolds number limit, we can neglect the nonlinear convective term from (2.10), leading to the unsteady Stokes equations,

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} - \nu \Delta \boldsymbol{u} + \nabla p &= \boldsymbol{f}, \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned} \tag{2.12}$$

Importantly, this is now a linear system of equations, greatly simplifying its solution.

We also consider the steady version of (2.12),

$$\begin{aligned} -\nu \Delta \boldsymbol{u} + \nabla p &= \boldsymbol{f}, \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned} \tag{2.13}$$

2.1.5 Convection-diffusion

We also consider the convection-diffusion equation,

$$\partial_t u + \nabla \cdot (\mathbf{v} u) - \nabla \cdot (\varepsilon(\mathbf{x}) \nabla u) = 0, \tag{2.14}$$

where $\boldsymbol{v}$ is a fixed velocity field, and ε is a variable diffusion coefficient. This equation is a simplified model for the more complicated dynamics of fluid flow. If $\|\boldsymbol{v}\| \ll \varepsilon$, then the equation is mostly diffusive and has properties similar to the heat equation. If $\|\boldsymbol{v}\| \gg \varepsilon$, then the equation is mostly convective and has properties similar to the advection equation. In particular, the Péclet number $Pe = \|\boldsymbol{v}\| L/\varepsilon$, where L is the characteristic length, can be used to classify the equation as either convection-dominated or diffusion-dominated.

2.2 Discontinuous Galerkin method

In this section, we formulate the discontinuous Galerkin (DG) method when applied to a system of conservation laws (2.1). In particular, we follow the compact discontinuous Galerkin (CDG) method [103] for handling diffusive fluxes in a compact manner. Throughout this chapter, we use the method of lines approach, where first we spatially discretize the PDE to create a semi-discrete system of equations, before applying a temporal discretization that yields a linear system to be solved at discrete time steps.

2.2.1 Function spaces

The spatial domain is denoted $\Omega \subset \mathbb{R}^d$ for $d = 2, 3$. In order to formulate the method, the spatial domain Ω is discretized using an unstructured mesh

$$\mathcal{T}_h = \left\{ K_i : \cup_{i=1}^{n_k} K_i = \Omega \right\}, \tag{2.15}$$

where each of the n_k elements has no overlap with any other. Each element K_i could be a simplex or tensor-product element, although our matrix-free operators introduced in Chapter 3 require tensor-product elements (quadrilaterals in 2D and hexahedra in 3D). For simplicity, we only consider tensor-product elements with $d = 3$ throughout this section. We take each element to be the image of T_i, an invertible transformation mapping from the reference element $\mathcal{R}$ to each element K_i. In the case of tensor-product elements, the reference element $\mathcal{R}$ would be the cube $[-1, 1]^d$.

We consider the space of polynomials of degree at most p in each variable,

$$\mathcal{Q}^p\left(\mathcal{R}\right) = \left\{ q : \mathcal{R} \to \mathbb{R}, q(x) = \sum_{i,j,k=0}^{p} c_i x^i c_j x^j c_k x^k \right\}, \tag{2.16}$$

noting this space has dimension $(p + 1)^d$. We define a nodal 1D basis as the Chebyshev points or $p + 1$ Legendre-Gauss-Lobatto points defined on the unit interval [64]. Either of these choices has a slowly growing Lebesque constant, reflecting the fact that functions interpolated on these points are accurately approximated by polynomials even as p increases, a necessary requirement of high-order methods. Tensor-product elements allow us to define the d-dimensional nodes as simply a tensor-product of 1D nodes. We will explore this tensor-product structure in greater detail in Chapter 3.

Now, with the fixed polynomial degree p, we define the function space

$$V_h(\mathcal{R}) = \mathcal{Q}^p\left(\mathcal{R}\right), \tag{2.17}$$

which induces a function space local to each element

$$V_h(K_i) = \left\{ v : K_i \to \mathbb{R} : v(x) = q\left(T_i^{-1}(\boldsymbol{x})\right), p \in V_h(\mathcal{R}) \right\}. \tag{2.18}$$

Therefore, for each basis function $\tilde{\phi}_j$ of $V_h(\mathcal{R})$, we have the transformed basis function ϕ_j of $V_h(K_i)$ defined by

$$\phi_j\left(\boldsymbol{x}\right) = \tilde{\phi}_j\left(T_i^{-1}\left(\boldsymbol{x}\right)\right). \tag{2.19}$$

Likewise, nodes on the reference element map to nodes on each element according to T_i.

Now we can introduce the broken finite element space

$$V_h = \left\{ v : \Omega \to \mathbb{R} : v|_{K_i} \in V_h(K_i) \ \forall K_i \in \mathcal{T}_h \right\}. \tag{2.20}$$

This translates to each v restricted to each element of the mesh existing in the corresponding local function space. Note that there is no continuity enforced between elements, leading to the descriptor "discontinuous" in discontinuous Galerkin method. We also define the same space for vector-valued functions. Taking n_c to be the number of components of the vector,

$$[V_h]^{n_c} = \left\{ \boldsymbol{v} = (v_1, v_2, \ldots, v_{n_c}) : \Omega \to \mathbb{R}^{n_c}, v_i \in V_h \ \forall 1 \leq i \leq n_c \right\}. \tag{2.21}$$

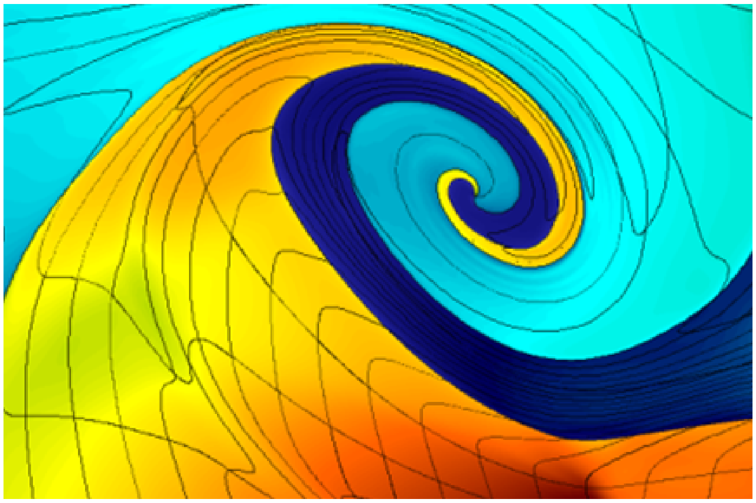


Figure 2.1: High-order $p = 7$ mesh accurately modeling a triple-point shock interaction [2]

The only requirement imposed on T_i so far has been that each mapping is invertible. For this thesis, we now choose the element mappings to be isoparametric, meaning that $T_i \in [V_h(\mathcal{R})]^d$. Since any function in this space can be fully defined by its values on the nodal basis points, the mapping is also defined by the location of the nodes on each element K_i. High-order elements allow for high-order mappings, leading to impressively complex meshes, such as the mesh in Figure 2.1, even with this mapping requirement. This isoparametric mesh is generated by a Lagrangian method at $p = 8$ propagating shock waves over multi-material regions leading to a so-called triple-point shock [2].

2.2.2 Weak formulation

We can now formulate the spatial discretization, noting that our specific treatment of the second-order terms follows the compact discontinuous Galerkin (CDG) method [103]. As a representative problem, we consider the discretization of a scalar conservation law (2.1) with $n = 1$ and appropriate boundary conditions enforced on $\partial\Omega$. Specifically, $u = \bar{u}_D$ on $\partial\Omega_D$ and $\partial_n u = \bar{u}_N$ on $\partial\Omega_N$, with the boundary $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$. Furthermore, we require the length of $\partial\Omega_D$ to be nonzero.

We introduce the additional variable $\boldsymbol{q} = \nabla u$, and rewrite the conservation law as a system of first-order equations,

$$\begin{aligned} \nabla u &= \boldsymbol{q} \\ \frac{\partial u}{\partial t} + \nabla \cdot F_c(u) - \nabla \cdot F_d(u, \boldsymbol{q}) &= 0 \end{aligned} \tag{2.22}$$

We look for numerical solutions $(u_h, \boldsymbol{q}_h) \in (V_h, [V_h]^d)$, multiply (2.22) by test functions

$(v, \boldsymbol{r}) \in (V_h, [V_h]^d)$, and integrate over Ω to obtain

$$\begin{aligned} \int_\Omega \boldsymbol{q}_h \cdot \boldsymbol{r} \; dx &= -\int_\Omega \nabla u_h \cdot \boldsymbol{r} \; dx \quad \forall \boldsymbol{r} \in [V_h]^d, \\ \int_\Omega \partial_t u_h v \; dx + \int_\Omega \nabla \cdot F_c(u) v \; dx &= \int_\Omega \nabla \cdot F_d \left(u_h, \boldsymbol{q}_h\right) v \; dx \quad \forall v \in V_h. \end{aligned} \tag{2.23}$$

Integrating by parts over each element, we derive the local weak formulation of (2.22):

$$\begin{aligned} \int_{K_i} \boldsymbol{q}_h \cdot \boldsymbol{r} \; dx &= -\int_K u_h \nabla \cdot \boldsymbol{r} \; dx + \int_{\partial K_i} \hat{u}_h \boldsymbol{r} \cdot \boldsymbol{n} \; ds, \\ \int_{K_i} \partial_t u_h v \; dx &= \int_{K_i} F_c(u_h) \cdot \nabla v \; dx - \int_{\partial K_i} v \hat{F}_c \cdot \boldsymbol{n} \; ds \\ &\quad - \int_{K_i} F_d(u_h, \boldsymbol{q_h}) \cdot \nabla v \; dx + \int_{\partial K_i} v \hat{F}_d \cdot \boldsymbol{n} \; ds. \end{aligned} \tag{2.24}$$

Since u_h is discontinuous across element interfaces, the flux functions F_c and F_d are undefined on ∂K_i. Therefore, u_h, F_c, and F_d on ∂K_i must be replaced with numerical fluxes $\hat{u}_h$, $\hat{F}_c$ and $\hat{F}_d$, respectively, that depend on the trace of u_h and $\boldsymbol{q}_h$ on both sides of ∂K_i, which are well-defined. The CDG formulation is only complete once we specify these numerical fluxes.

For two neighboring elements K^+ and K^- of $\mathcal{T}_h$, we denote their common face $e = \partial K^+ \cap \partial K^-$. In particular, we adopt the notation that the normal unit vector $\boldsymbol{n}^-$ points outward from the element associated to the values $(\cdot)^-$. We denote the unit normals to $\partial K^\pm$ as $\boldsymbol{n}^\pm$ at any point on face e. The average and jump operators of v are given as

$$\{v\} = \left(v^+ + v^-\right)/2, \qquad [v] = v^+ \boldsymbol{n}^+ + v^- \boldsymbol{n}^-. \tag{2.25}$$

Similarly, the average and jump operators of the vector-valued function $\boldsymbol{r}$ are given as

$$\{\boldsymbol{r}\} = \left(\boldsymbol{r}^+ + \boldsymbol{r}^-\right)/2, \qquad [\boldsymbol{r}] = \boldsymbol{r}^+ \cdot \boldsymbol{n}^+ + \boldsymbol{r}^- \cdot \boldsymbol{n}^-. \tag{2.26}$$

Note that this defines the jump of a scalar quantity as a vector and the jump of a vector quantity as a scalar. We define Γ to be the union of all interior faces of the triangulation $\mathcal{T}_h$. Then we obtain the following global weak formulation: find $(u_h, \boldsymbol{q}_h) \in (V_h, [V_h]^d)$ such that,

$$\begin{aligned} \int_\Omega \boldsymbol{q}_h \cdot \boldsymbol{r} \; dx &= -\int_\Omega u_h \nabla_h \cdot \boldsymbol{r} \; dx + \int_\Gamma \hat{u}_h [\boldsymbol{r}] \; ds + \int_{\partial\Omega} \hat{u}_h \boldsymbol{r} \cdot \boldsymbol{n} \; ds, \\ \int_\Omega \partial_t u_h v \; dx &= \int_\Omega F_c(u_h) \cdot \nabla_h v \; dx - \int_\Gamma \hat{F}_c\left(u_h^-, u_h^+\right) \cdot [v] \; ds - \int_{\partial\Omega} v \hat{F}_c \cdot \boldsymbol{n} \; ds \\ &\quad - \int_\Omega F_d\left(u_h, \boldsymbol{q}_h\right) \cdot \nabla_h v \; dx + \int_\Gamma \hat{F}_d\left(u_h^-, u_h^+, \boldsymbol{q}_h^-, \boldsymbol{q}_h^+\right) \cdot [v] \; ds + \int_{\partial\Omega} v \hat{F}_d \cdot \boldsymbol{n} \; ds, \end{aligned} \tag{2.27}$$

for all $\boldsymbol{r} \in [V_h]^d$ and $v \in V_h$. Here, we introduce ∇_h as the broken gradient operator, or the gradient operator computed element-wise, noting that it is only well-defined within each element and not on the element boundaries.

The convective flux function $\hat{F}_c(u_h^-, u_h^+)$ couples neighboring elements and is chosen to stabilize the discontinuities inherent to the DG method. On the boundary, $\hat{F}_c$ weakly enforces the Dirichlet boundary condition on $\partial\Omega_D$ and is equal to 0 on $\partial\Omega_N$. Internally on Γ, $\hat{F}_c$ must be consistent, $\hat{F}_c(u, u) = u$, ensuring the amount of u flowing out of one element equals the amount of u flowing into the other element. To stabilize the DG method, $\hat{F}_c$ is obtained by finding an exact or approximate solution to the one-dimensional Riemann problem in the normal direction of the face, with discontinuous initial data given by u_h^- and u_h^+.

A simple choice of numerical flux function is the local Lax-Friedrichs flux [81], given by

$$\hat{F}_c(u_h^-, u_h^+) = \{F_c(u_h)\} + \frac{\alpha}{2}[u_h], \tag{2.28}$$

where α is the maximum absolute eigenvalue of the two Jacobian matrices $B(u_h^-)$ and $B(u_h^+)$, for

$$B(u) = \frac{\partial F_c(u)}{\partial u} \cdot \boldsymbol{n}. \tag{2.29}$$

Thus, α is equivalent to the maximum speed achieved by the flow at this interface.

A more sophisticated numerical flux function uses the Roe approximate Reimann solver [110, 111]. This numerical flux function is valid for the Navier-Stokes equations (2.2), exactly solving the linearized Riemann problem around the Roe averaged state given by

$$\begin{aligned} \hat{\boldsymbol{u}}_h &= \frac{\sqrt{\rho_h^-}\boldsymbol{u}_h^- + \sqrt{\rho_h^+}\boldsymbol{u}_h^+}{\sqrt{\rho_h^-} + \sqrt{\rho_h^+}}, \\ \hat{H}_h &= \frac{\sqrt{\rho_h^-}H_h^- + \sqrt{\rho_h^+}H_h^+}{\sqrt{\rho_h^-} + \sqrt{\rho_h^+}}. \end{aligned} \tag{2.30}$$

This linear approximation u^* is an efficient approximation to the solution of the nonlinear, true Riemann problem. The Roe numerical flux function is then given by $\hat{F}_c(u^*)$.

2.2.3 CDG fluxes

Much research has been done to define specific flux choices for elliptic and parabolic equations, including the interior penalty method [47], first and second methods of Bassi and Rebay [9, 10], local DG method [39], and compact DG method [103]. While a comprehensive study is presented in [4], here we continue our presentation of CDG and only briefly note some of the important properties of this choice.

We first seek to eliminate the auxiliary variable $\boldsymbol{q}_h$ using the integration by parts formula,

$$-\int_\Omega v\nabla_h\cdot\boldsymbol{r}\;dx=\int_\Omega \boldsymbol{r}\cdot\nabla_h v\;dx-\int_\Gamma\left([v]\cdot\{\boldsymbol{r}\}+\{v\}\,[\boldsymbol{r}]\right)\;ds-\int_{\partial\Omega} v\boldsymbol{r}\cdot\boldsymbol{n}\;ds,\tag{2.31}$$

which is valid for all $\boldsymbol{r}\in[V_h]^d$ and $v\in V_h$. Plugging (2.31) in for the first term on the right-hand side of the first equation of (2.27) yields,

$$\int_\Omega \boldsymbol{q}_h\cdot\boldsymbol{r}\;dx=\int_\Omega \boldsymbol{r}\cdot\nabla_h u_h\;dx-\int_\Gamma\left([u_h]\cdot\{\boldsymbol{r}\}-\{\hat{u}_h-u_h\}\,[\boldsymbol{r}]\right)\;ds+\int_{\partial\Omega}(\hat{u}_h-u_h)\boldsymbol{r}\cdot\boldsymbol{n}\;ds.\tag{2.32}$$

Next we define the face-wise lifting operators $\ell^e_{\boldsymbol{r}}:[L^2(e)]^d\to[V_h]^d$, $\ell^e_v:L^2(e)\to[V_h]^d$, and $\ell^e_D:L^2(e)\to[V_h]^d$ as

$$\begin{aligned}\int_\Omega \ell^e_{\boldsymbol{r}}(\boldsymbol{q})\cdot\boldsymbol{r}&=-\int_e \boldsymbol{q}\cdot\{\boldsymbol{r}\}\;\;ds,\\ \int_\Omega \ell^e_v(v)\cdot\boldsymbol{r}&=-\int_e v\,[\boldsymbol{r}]\;\;ds,\\ \int_\Omega \ell^e_D(v)\cdot\boldsymbol{r}&=-\int_e v\boldsymbol{r}\cdot\boldsymbol{n}\;ds.\end{aligned}\tag{2.33}$$

Summing over all relevant faces yields the element-wise lifting operators $\ell_{\boldsymbol{r}}:[L^2(\Gamma)]^d\to[V_h]^d$, $\ell_v:L^2(\Gamma)\to[V_h]^d$, $\ell_D:L^2(\partial\Omega_D)\to[V_h]^d$, simply defined as

$$\begin{aligned}\ell_{\boldsymbol{r}}(\boldsymbol{q})&=\sum_{e\in\Gamma}\ell^e_{\boldsymbol{r}}(\boldsymbol{q}),\\ \ell_v(v)&=\sum_{e\in\Gamma}\ell^e_v(v),\\ \ell_D(v)&=\sum_{e\in\partial\Omega_D}\ell^e_D(v).\end{aligned}\tag{2.34}$$

With this machinery in place, we can explicitly define $\boldsymbol{q}_h$ from (2.32) as

$$\boldsymbol{q}_h=\nabla_h u_h+\ell_{\boldsymbol{r}}\left([u_h]\right)+\ell_v\left(\{u_h-\hat{u}_h\}\right)+\ell_D\left(u_h-\hat{u}_h\right).\tag{2.35}$$

We now define the numerical inter-element flux $\hat{u}_h$ so that $\boldsymbol{q}_h$ can be fully eliminated. Both LDG and CDG define $\hat{u}_h$ as

$$\begin{aligned}\hat{u}_h=\{u_h\}-\boldsymbol{C}_{12}\cdot[u_h]&\quad\text{on }\Gamma,\\ \hat{u}_h=\bar{u}_D&\quad\text{on }\partial\Omega_D,\\ \hat{u}_h=u_h&\quad\text{on }\partial\Omega_N.\end{aligned}\tag{2.36}$$

$\boldsymbol{C}_{12}$ is a vector which is determined for each interior face according to

$$\boldsymbol{C}_{12}=\frac{1}{2}\left(S^{K^-}_{K^+}\boldsymbol{n}^++S^{K^+}_{K^-}\boldsymbol{n}^-\right).\tag{2.37}$$

$S_{K^+}^{K^-} \in \{0, 1\}$ is a switch which is defined for each element face, satisfying

$$S_{K^+}^{K^-} + S_{K^-}^{K^+} = 1. \tag{2.38}$$

In particular, this means that $\hat{u}_h$ simplifies to either u_h^+ or u_h^-, depending on the particular switch defined on the face between K^+ and K^-.

The key development of CDG is the careful choice of $\hat{F}_d$. CDG defines $\hat{F}_d$ as

$$\begin{aligned} \hat{F}_d\left(u_h^-, u_h^+, \boldsymbol{q}_h^-, \boldsymbol{q}_h^+\right) &= F_d\left(u_h^-, \{\boldsymbol{q}_h^e\} - C_{11}\left[u_h\right] + \boldsymbol{C}_{12}\left[\boldsymbol{q}_h^e\right]\right) && \text{on } \Gamma, \\ \hat{F}_d &= F_d\left(u_h, \boldsymbol{q}_h^e - C_{11}(u_h - \bar{u}_D)\right)\boldsymbol{n} && \text{on } \partial\Omega_D, \\ \hat{F}_d &= F_d\left(u_h, \bar{u}_N\right)\boldsymbol{n} && \text{on } \partial\Omega_N. \end{aligned} \tag{2.39}$$

The parameter $C_{11} \geq 0$ is chosen to stabilize the method, and can be thought of as adding artificial viscosity. The face-wise choice of auxiliary variable for the numerical flux function is defined as

$$\boldsymbol{q}_h^e = \nabla_h u_h + \ell_{\boldsymbol{r}}^e\left([u_h]\right) + \ell_v^e\left(\boldsymbol{C}_{12} \cdot [u_h]\right) - \ell_D^e\left(\bar{u}_D - u_h\right). \tag{2.40}$$

First we note that the auxiliary variable used in $\hat{F}_d\left(u_h^-, u_h^+, \boldsymbol{q}_h^-, \boldsymbol{q}_h^+\right)$ is chosen to be exactly the opposite of the switch used to compute $\hat{u}_h$ in (2.36). Importantly, we also note that the support of $\boldsymbol{q}_h^e$ is localized to the current element and the element neighboring face e. Thus, the stencil of the CDG operator is compact, in that the residual corresponding to a test function whose support lies entirely in a given element depends only on the value of the trial function in the immediately adjacent elements. This compactness property is crucial for the preconditioners developed in Chapter 5.

This specific choice of fluxes particularizes (2.35) to

$$\boldsymbol{q}_h = \nabla_h u_h + \ell_{\boldsymbol{r}}\left([u_h]\right) + \ell_v\left(\boldsymbol{C}_{12} \cdot [u_h]\right) - \ell_D\left(\bar{u}_D - u_h\right), \tag{2.41}$$

and allows us to fully compute $\boldsymbol{q}_h$ locally from u_h. Thus, our global weak formulation (2.27) is reduced to just the second equation combined with a local solve of $\boldsymbol{q}_h$.

2.2.4 Semi-discrete system

After applying our DG discretization, we obtain a system of equations for u_h (2.27). Let $N = (p+1)^d n_k$ denote the dimension of the space V_h. Any function $u_h \in V_h$ can be expanded in terms of the basis functions $\{\phi_i\}_{i=1}^N$. By construction of V_h, these ϕ_i have compact support on one element K_i, satisfying $\phi_i(x_j) = \delta_{ij}$ at the nodal points x_j of K_i, and 0 outside of K_i. These ϕ_i at various p are presented for the reference element in Figure 2.2. Therefore,

$$u_h(\boldsymbol{x}, t) = \sum_{j=1}^{N} u_j(t)\phi_j(\boldsymbol{x}) \tag{2.42}$$

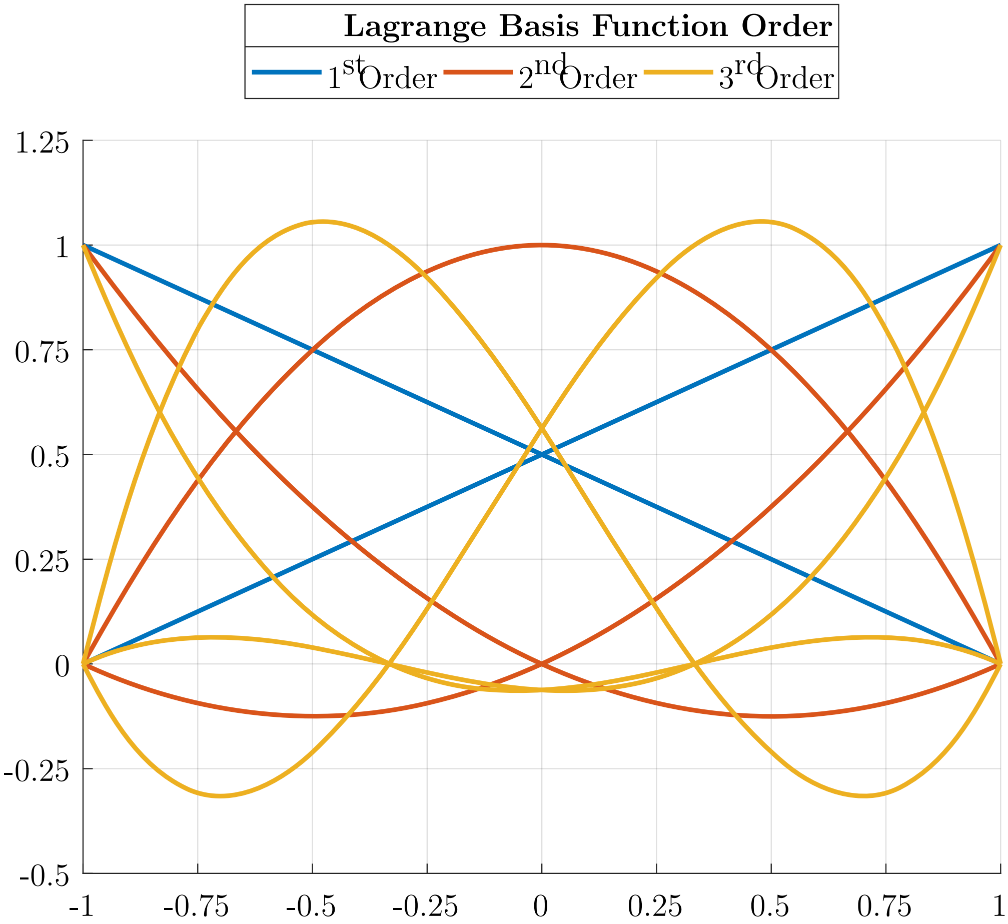


Figure 2.2: Basis functions $\{\phi_i\}_{i=1}^{p+1}$ for $p \in \{1, 2, 3\}$ on the 1D reference element $\mathcal{R} = [-1, 1]$.

where we abuse notation by re-using u_j to refer to the coefficients of this expansion as well as the function itself evaluated at the nodal point $\boldsymbol{x}_j$ since $u(\boldsymbol{x}_j) = u_j$. Now that ϕ_i form a basis of V_h, ensuring (2.27) holds for all $v \in V_h$ is equivalent to ensuring it holds for each ϕ_i. Thus, for all $1 \leq i \leq N$,

$$\sum_{j=1}^{N} \partial_t u_j \int_\Omega \phi_j \phi_i \, dx = \int_\Omega F_c \left(\sum_{j=1}^{N} u_j \phi_j \right) \cdot \nabla_h \phi_i \, dx - \int_\Gamma \hat{F}_c \cdot [\phi_i] \, ds - \int_{\partial\Omega} \phi_i \hat{F}_c \cdot \boldsymbol{n} \, ds$$
$$- \int_\Omega F_d \left(\sum_{j=1}^{N} u_j \phi_j, \boldsymbol{q}_h \right) \cdot \nabla_h v \, dx + \int_\Gamma \hat{F}_d \cdot [\phi_i] \, ds + \int_{\partial\Omega} \phi_i \hat{F}_d \cdot \boldsymbol{n} \, ds. \tag{2.43}$$

Defining $M \in \mathbb{R}^{N \times N}$ to be the mass matrix,

$$M_{ij} = \int_\Omega \phi_i \phi_j \, dx, \tag{2.44}$$

and $\boldsymbol{r} : \mathbb{R}^N \to \mathbb{R}^N$ to be the residual vector,

$$\begin{aligned}(\boldsymbol{r}(\boldsymbol{u}))_i = \int_\Omega F_c\left(\sum_{j=1}^N u_j\phi_j\right)\cdot\nabla_h\phi_i\ dx - \int_\Gamma \hat{F}_c\cdot[\phi_i]\ ds - \int_{\partial\Omega}\phi_i\hat{F}_c\cdot\boldsymbol{n}\ ds \\ - \int_\Omega F_d\left(\sum_{j=1}^N u_j\phi_j, \boldsymbol{q}_h\right)\cdot\nabla_h v\ dx + \int_\Gamma \hat{F}_d\cdot[\phi_i]\ ds + \int_{\partial\Omega}\phi_i\hat{F}_d\cdot\boldsymbol{n}\ ds,\end{aligned} \tag{2.45}$$

we obtain a system of coupled ordinary differential equations (ODEs) of the form

$$M\dot{\boldsymbol{u}} = \boldsymbol{r}\left(\boldsymbol{u}\right). \tag{2.46}$$

This system of ODEs is known as the semi-discrete system of equations corresponding to the DG formulation. This is because $\boldsymbol{u}(t)$ is a vector of time-dependent degrees of freedom associated with the nodal basis representation of u_h.

Our mass matrix is by construction symmetric positive definite. By the element-wise compact support of each ϕ_i, M furthermore has an element-wise block-diagonal structure. Thus, matrix-multiplication by M and its inverse are both element-wise local and computationally efficient.

2.3 Implicit time integration

We solve the semi-discrete system (2.46) using implicit time integration techniques. In particular, we focus on diagonally implicit Runge-Kutta (DIRK) schemes [104, 1, 68], although we will also briefly utilize backward differentiation formulae (BDF) [62, 42, 68] in Chapter 4. Both of these methods require the solution of s nonlinear systems of equations of size N.

2.3.1 Diagonally implicit Runge-Kutta schemes

We consider the well-posed initial value problem defined by (2.46) and an appropriate initial condition $\boldsymbol{u}(0) = \boldsymbol{u}_0$.

Let $\boldsymbol{u}^n$ denote the known solution at time t^n. A general s-stage Runge-Kutta scheme to approximate the solution at time $t^{n+1} = t^n + \Delta t$ can be written as

$$\begin{aligned}M\boldsymbol{k}_i^n &= \boldsymbol{r}\left(t^n + \Delta t c_i, \boldsymbol{u}^n + \Delta t\sum_{j=1}^s a_{ij}\boldsymbol{k}_j^n\right), \\ \boldsymbol{u}^{n+1} &= \boldsymbol{u}^n + \Delta t\sum_{i=1}^s b_i\boldsymbol{k}_i^n,\end{aligned} \tag{2.47}$$

where the coefficients a_{ij}, b_i, and c_i can be expressed compactly in the form of the *Butcher tableau*,

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} = \begin{array}{c|c} \boldsymbol{c} & A \\ \hline & \boldsymbol{b}^T \end{array}. \tag{2.48}$$

A Runge-Kutta scheme is diagonally implicit if its Butcher matrix A is lower triangular, allowing for the solution of the system of equations (2.47) through a forward-substitution procedure. For each stage $1 \leq i \leq s$, $\boldsymbol{k}_i^n$ is implicitly defined in a DIRK scheme by

$$M\boldsymbol{k}_i^n - \boldsymbol{r}\left(t^n + \Delta t c_i, \boldsymbol{u}^n + \Delta t\left(\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j^n + a_{ii}\boldsymbol{k}_i^n\right)\right) = 0. \tag{2.49}$$

Each stage therefore requires the solution of a backward Euler-type system. Thus, rather than solve one $s \times N$ system of nonlinear equations, DIRK schemes require the solution of s nonlinear systems of equations. Each stage is solved by applying Newton's method [72] to this nonlinear system. The Newton update $\Delta\boldsymbol{x} \in \mathbb{R}^N$ defined by $\Delta\boldsymbol{x} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$ is the solution of the linear system,

$$\frac{\partial\boldsymbol{N}\left(\boldsymbol{x}^{(k)}\right)}{\partial\boldsymbol{x}}\Delta\boldsymbol{x} = -\boldsymbol{N}\left(\boldsymbol{x}^{(k)}\right), \tag{2.50}$$

for $\boldsymbol{N}$ defined as the left hand side of (2.49) as a function of $\boldsymbol{k}_i^n$. It is well known that that the Newton approximation $\boldsymbol{x}^{(k)}$ converges to the true $\boldsymbol{k}_i^n$ quadratically. We choose $\boldsymbol{x}^{(0)} = \boldsymbol{k}_{i-1}^n$ as the initial guess. Thus, (2.50) expands to

$$\left(M - a_{ii}\Delta t J\right)\Delta\boldsymbol{x} = M\boldsymbol{x}^{(k)} - \boldsymbol{r}\left(t^n + \Delta t c_i, \boldsymbol{u}^n + \Delta t\left(\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j^n + a_{ii}\boldsymbol{x}^{(k)}\right)\right), \tag{2.51}$$

where J is the $N \times N$ Jacobian matrix,

$$J = \frac{\partial\boldsymbol{r}}{\partial\boldsymbol{u}}\left(t^n + \Delta t c_i, \boldsymbol{u}^n + \Delta t\left(\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j^n + a_{ii}\boldsymbol{x}^{(k)}\right)\right). \tag{2.52}$$

This formulation shows how fundamental (2.51) is to the overall solution of the original PDE. If the assembly and solution of this linear system is performant, then Newton's method is performant, the DIRK scheme is performant, and ultimately the time to solution is minimized. Throughout this thesis, we will focus on solving this linear system (2.51) efficiently, so we simplify it to

$$A\boldsymbol{x} = \left(M - \Delta t J\right)\boldsymbol{x} = \boldsymbol{b}. \tag{2.53}$$

That is, we define $\boldsymbol{x}$ and $\boldsymbol{b}$ appropriately and without loss of generality take $a_{ii} = 1$ by scaling Δt by this positive constant when necessary. We use A to denote the linear operator in (2.53).

The specific DIRK schemes that we use in this thesis are provided in Appendix A.1. Of particular interest are the so-called singly DIRK schemes, which use identical diagonal coefficients a_{ii}. These schemes allow for reuse of preconditioners, since each stage requires the solution of a backward Euler system with the same time step.

Finally, we note that the application of DIRK schemes to the incompressible systems (2.10) and (2.12) is complicated by the fact that there is no temporal evolution equation corresponding to the pressure. We will further discuss their temporal discretization in combination with specific solvers in Chaper 4.

2.4 Krylov subspace solvers

In order to solve the linear systems described above, we make use of preconditioned Krylov subspace methods, such as the conjugate gradient (CG) and generalized minimal residual (GMRES) methods [116]. Beyond being prevalent for their robustness, these iterative solvers are also a natural choice for matrix-free solvers, since they only require the action of the operator, which we compute using the matrix-free algorithms we will describe in Chapter 3, and the evaluation of a preconditioner.

We briefly describe these well-known Krylov subspace methods here, following their construction from [116]. Suppose we wish to solve the $N \times N$ linear system,

$$A\boldsymbol{x} = \boldsymbol{b}. \tag{2.54}$$

We consider $\boldsymbol{x}_0$ as the initial guess for the true solution $\boldsymbol{x}_*$ and define the initial residual $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$. Krylov subspace methods are constructed as an iterative process whose j^{th} iterate $\boldsymbol{x}_j$ satisfies

$$\boldsymbol{x}_j \in \boldsymbol{x}_0 + \mathcal{K}_j(A, \boldsymbol{r}_0), \tag{2.55}$$

where $\mathcal{K}_j(A, \boldsymbol{r}_0)$ is the j^{th} Krylov subspace generated by A and $\boldsymbol{r}_0$ for all $1 \le j \le N$, defined by

$$\mathcal{K}_j(A, \boldsymbol{r}_0) = \operatorname{span}\left\{\boldsymbol{r}_0, A\boldsymbol{r}_0, A^2\boldsymbol{r}_0, \ldots, A^{j-1}\boldsymbol{r}_0\right\}. \tag{2.56}$$

2.4.1 Conjugate gradient method

The conjugate gradient (CG) method is one of the best known iterative techniques for solving sparse symmetric positive-definite linear systems [63, 120]. Each iteration, CG orthogonally projects onto the Krylov subspace $\mathcal{K}_j(A, \boldsymbol{r}_0)$, providing the approximation $\boldsymbol{x}_j$ that minimizes the A-norm of the error within this subspace, all while taking advantage of the symmetry inherent in A. We present the CG method in Algorithm 1.

Algorithm 1 Conjugate gradient method

```
function CG($A$, $\boldsymbol{b}$, $\boldsymbol{x}_0$)
    $\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - A\boldsymbol{x}_0$
    $\boldsymbol{p}_0 \leftarrow \boldsymbol{r}_0$
    for $j \leftarrow 0, 1, \ldots$ do
        $\alpha_j \leftarrow \dfrac{(\boldsymbol{r}_j, \boldsymbol{r}_j)}{(A\boldsymbol{p}_j, \boldsymbol{p}_j)}$
        $\boldsymbol{x}_{j+1} \leftarrow \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j$
        $\boldsymbol{r}_{j+1} \leftarrow \boldsymbol{r}_j - \alpha_j A\boldsymbol{p}_j$
        $\beta_j \leftarrow \dfrac{(\boldsymbol{r}_{j+1}, \boldsymbol{r}_{j+1})}{(\boldsymbol{r}_j, \boldsymbol{r}_j)}$
        $\boldsymbol{p}_{j+1} \leftarrow \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{p}_j$
    end for
    return $\boldsymbol{x}_{j+1}$
end function
```

It can be shown that the approximation $\boldsymbol{x}_m$ generated by CG after m iterations satisfies

$$\|\boldsymbol{x}_* - \boldsymbol{x}_m\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^m \|\boldsymbol{x}_* - \boldsymbol{x}_0\|_A \, , \tag{2.57}$$

where κ is the spectral condition number of A,

$$\kappa = \lambda_{max} / \lambda_{min}. \tag{2.58}$$

2.4.2 Generalized minimal residual method

The generalized minimum residual (GMRES) method is an iterative method for solving indefinite nonsymmetric linear systems [115]. Each iteration, GMRES approximates the solution by the vector in the Krylov subspace $\mathcal{K}_j(A, \boldsymbol{r}_0)$ with minimum residual. We present the GMRES method in Algorithm 2.

One can show that $h_{j+1,j} = 0$ if and only if the approximate solution $\boldsymbol{x}_j$ is exact. Thus, one can stop GMRES early if $h_{j+1,j} = 0$. In general, the residual of the j^{th} iteration is bounded by

$$\|\boldsymbol{r}_j\|_2 \leq \min_{q \in \mathcal{P}_1^j} \|q(A)\|_2 \, \|\boldsymbol{r}_0\|_2 \, , \tag{2.59}$$

where $\mathcal{P}_1^j$ is the set of polynomials of at most degree j for which $q(0) = 1$. Thus, the convergence of GMRES is guided by the eigenvalues of the underlying matrix A. Indeed, the existence of clustered eigenvalues speeds up the convergence of the method [14].

Moreover, GMRES has been extended to create flexible GMRES (FGMRES) [114]. This iterative method allows for the use of different preconditioners throughout the iterations.

Algorithm 2 Generalized minimum residual method

```
function GMRES($A$, $\boldsymbol{b}$, $\boldsymbol{x}_0$)
    $\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - A\boldsymbol{x}_0$
    $\beta \leftarrow \|\boldsymbol{r}_0\|_2$
    $\boldsymbol{v}_1 \leftarrow \boldsymbol{r}_0/\beta$
    for $j \leftarrow 1, 2, \dots, m$ do
        $\boldsymbol{w}_j \leftarrow A\boldsymbol{v}_j$
        for $i \leftarrow 1, 2, \dots, j$ do
            $h_{ij} \leftarrow (\boldsymbol{w}_j, \boldsymbol{v}_i)$
            $\boldsymbol{w}_j \leftarrow \boldsymbol{w}_j - h_{ij}\boldsymbol{v}_i$
        end for
        $h_{j+1,j} \leftarrow \|\boldsymbol{w}_j\|_2$
        $\boldsymbol{v}_{j+1} \leftarrow \boldsymbol{w}_j/h_{j+1,j}$
    end for
    $\boldsymbol{y}_m \leftarrow \operatorname{argmin}_{\boldsymbol{y}} \|\beta\boldsymbol{e}_1 - H_m\boldsymbol{y}\|_2$
    $\boldsymbol{x}_m \leftarrow \boldsymbol{x}_0 + V_m\boldsymbol{y}_m$
    return $x_m$
end function
```

We will use FGMRES when our desired preconditioners are themselves iterative methods, such as CG or GMRES. In particular, the iterative subregion correction preconditioners developed in Chapter 5 are iterative methods, and therefore can only be used in conjunction with FGMRES.

2.4.3 Preconditioning

Since the convergence rates of these Krylov subspace methods are governed by the eigenvalues of the underlying linear operator, it is often beneficial to transform the original system (2.54) to achieve more favorable convergence rates. Rather than solve the original system, we instead consider the left-preconditioned system

$$P^{-1}A\boldsymbol{x} = P^{-1}\boldsymbol{b}, \tag{2.60}$$

or right-preconditioned system

$$\begin{aligned} AP^{-1}\boldsymbol{y} &= \boldsymbol{b}, \\ \boldsymbol{x} &= P^{-1}\boldsymbol{y}. \end{aligned} \tag{2.61}$$

For these transformed systems, the convergence of the Krylov subspace methods is instead controlled by the eigenvalues of $P^{-1}A$ or AP^{-1}. Thus, a good preconditioner P^{-1} approximates A^{-1}, so that the transformed eigenvalues are clustered around 1. However, since the application of the preconditioner must be applied at each iteration of the solver, it also must be cheap. Moreover, in the case of the matrix-free solvers we develop in Chapter 4, the A matrix is not explicitly assembled, so we must also create a preconditioner without access to these underlying entries. Throughout this thesis, we will develop novel, efficient preconditioners with the goal of minimizing the total number of iterations required by Krylov subspace methods to converge to a specified tolerance.

Chapter 3

Matrix-free high-order operators with GPU acceleration

The high-order methods we described in Chapter 2 ultimately require the efficient construction and solution of the linear system (2.53). Therefore, the matrix-vector products $\boldsymbol{Ax}$ and those composing $\boldsymbol{b}$ are important computational kernels to efficiently implement. Furthermore, a key benefit of high-order methods is their high arithmetic intensity, or operations per memory access, which is particularly relevant to the matrix-vector product kernel. Programs run on graphics processing units (GPUs) must minimize memory movement to reap the benefits of the GPU architecture, so arithmetic intensity is a key concern [74, 131, 25]. In this chapter, we propose an efficient implementation of these high-order methods on GPUs and GPU-accelerated architectures of current and future supercomputers.

Unfortunately, the benefits of high-order methods do not immediately imply that running increasingly higher order simulations for a fixed problem size will result in more efficient simulations. In general, Galerkin finite element methods (FEM) couple all degrees of freedom (DoFs) within each mesh element. Therefore, the memory required to store the resulting system matrices grows quadratically with the number of DoFs per element (i.e. $\mathcal{O}(p^{2d})$ in d spatial dimensions, where p is the polynomial degree). Furthermore, the naive assembly of the system matrix requires $\mathcal{O}(p^{3d})$ operations, although we will present sum-factorization techniques that lower this to $\mathcal{O}(p^{2d+1})$ [78, 73]. Thus, traditional matrix-based approaches are impractical for use with high polynomial degrees, both in terms of computational cost and memory requirements.

Instead, matrix-vector products can be replaced with on-the-fly evaluations of the discretized differential operators in a matrix-free manner. Combined with sum-factorization techniques on tensor-product elements originally developed in the spectral element community [94, 96], evaluation of these matrix-free operators can be performed in $\mathcal{O}(p^{d+1})$ operations and $\mathcal{O}(p^d)$ memory [100]. Matrix-free evaluations with sum factorization have been shown to outperform sparse matrix-vector products with $p \geq 2$ due to bandwidth bounds on modern architectures [77]. Furthermore, optimized implementations of these operators achieve near peak performance on modern GPUs [25, 125], making matrix-free high-order operators a desirable choice for performant implementations. In this chapter, we explore and define these sum-factorization techniques.

3.1 Towards element-wise kernels

By the definition of V_h for DG methods in (2.20), the basis functions $\{\phi_i\}_{i=1}^N$ spanning V_h each have compact support on one element. Therefore, the mass matrix (2.44) and system matrices composing the residual (2.45) naturally have an element-wise block-diagonal structure. In particular, this means that an efficient implementation of these methods can directly focus on element-wise local matrix multiplications instead.

High-order FEM, however, does not share this direct connection to local matrices. For this chapter, we consider the equations governing incompressible fluid flow, although these concepts directly apply to compressible flow as well. That is, we consider the steady (2.11) and unsteady incompressible Navier-Stokes equations (2.10), as well as the steady (2.13) and unsteady Stokes equations (2.12), all with Dirichlet boundary conditions.

We define the following finite element function spaces on our mesh $\mathcal{T}_h$:

$$\begin{aligned} V_p &= \left\{ \boldsymbol{v} \in \left(H^1(\Omega)\right)^d \ \middle|\ \boldsymbol{v}(K_i) \in \left(\mathcal{Q}^p(K_i)\right)^d \ \forall K \in \mathcal{T}_h \right\} \\ P_q &= \left\{ s \in H^1(\Omega) \ \middle|\ s(K_i) \in \mathcal{Q}^p(K_i) \ \forall K_i \in \mathcal{T}_h \right\} \end{aligned} \tag{3.1}$$

We again use the Legendre-Gauss-Lobatto tensor-product nodal points and form bases on each of the spaces V_p and P_q. The high-order finite element formulation for unsteady Stokes (2.12) is: find a velocity-pressure pair $(\boldsymbol{u}, p) \in (V_p, P_q)$ such that

$$\begin{aligned} \left(\frac{\partial \boldsymbol{u}}{\partial t}, \boldsymbol{v}\right) + (\nu \nabla \boldsymbol{u}, \nabla \boldsymbol{v}) + (\nabla p, \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in V_p, \\ -(\nabla \cdot \boldsymbol{u}, s) &= 0 \qquad\qquad \forall s \in P_q. \end{aligned} \tag{3.2}$$

Expanding out $\boldsymbol{u}$, $\boldsymbol{v}$, p, and s in terms of the bases for their respective spaces, we obtain the semi-discrete Stokes problem:

$$\begin{aligned} \boldsymbol{M}\dot{\boldsymbol{u}} + \boldsymbol{L}\boldsymbol{u} + \boldsymbol{G}p &= \boldsymbol{f}, \\ -Du &= 0, \end{aligned} \tag{3.3}$$

where $\boldsymbol{u}$ and p are now reused to represent the vectors of coefficients of the high-order polynomial basis functions approximating their continuous counterparts. $\boldsymbol{M}$ is the vector mass matrix, $\boldsymbol{L}$ is the vector stiffness matrix, $\boldsymbol{G}$ is the gradient operator, and D is the divergence operator with the following definitions:

$$\boldsymbol{M}_{ij} = \int_\Omega \boldsymbol{\phi}_i \boldsymbol{\phi}_j \; dx, \tag{3.4}$$

$$\boldsymbol{L}_{ij} = \int_\Omega \nu \nabla \boldsymbol{\phi}_i \cdot \nabla \boldsymbol{\phi}_j \; dx, \tag{3.5}$$

$$\boldsymbol{G}_{ij} = \int_\Omega \boldsymbol{\phi}_i \cdot \nabla \psi_j \; dx, \tag{3.6}$$

$$D_{ij} = \int_\Omega \psi_i \nabla \cdot \boldsymbol{\phi}_j \; dx. \tag{3.7}$$

The bases $\{\boldsymbol{\phi}_i\}_{i=1}^{n_v}$ and $\{\psi_j\}_{j=1}^{n_p}$ span the velocity space V_p and pressure space P_q, respectively. In the steady Stokes problem we take $\dot{\boldsymbol{u}} = 0$, so (3.3) reduces to the linear system

$$\begin{bmatrix} \boldsymbol{L} & \boldsymbol{G} \\ -D & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ p \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ 0 \end{bmatrix}. \tag{3.8}$$

A similar treatment of the incompressible Navier-Stokes equations (2.10) yields the semi-discrete problem:

$$\begin{aligned} \boldsymbol{M}\dot{\boldsymbol{u}} + \boldsymbol{L}\boldsymbol{u} + \boldsymbol{N}(\boldsymbol{u}) + \boldsymbol{G}p &= \boldsymbol{f}, \\ -Du &= 0, \end{aligned} \tag{3.9}$$

where $\boldsymbol{N}(\boldsymbol{u})$ is the discretized nonlinear vector-convection term defined by

$$\boldsymbol{N}(\boldsymbol{u})_i = \int_\Omega \boldsymbol{u}^T (\boldsymbol{\Phi} \cdot \nabla) \boldsymbol{\Phi} \boldsymbol{u} \boldsymbol{\phi}_i \; dx. \tag{3.10}$$

Equation (3.10) uses the notation $\boldsymbol{\Phi}$ to represent the tensor of all $\boldsymbol{\phi}_i$, so that $(\boldsymbol{\Phi} \cdot \nabla)\boldsymbol{\Phi}$ can be viewed as a matrix of size $n_v \times n_v$ where each entry is the vector

$$[(\boldsymbol{\Phi} \cdot \nabla)\boldsymbol{\Phi}]_{ij} = (\boldsymbol{\phi}_i \cdot \nabla)\boldsymbol{\phi}_j. \tag{3.11}$$

When solving the steady (3.8) and unsteady Stokes (3.3) problems, we use the Taylor-Hood finite element space, $(\boldsymbol{u}, p) \in (V_p, P_{p-1})$, which achieves optimal convergence rates and is stable for orders $p \geq 2$ [21]. When solving the incompressible Navier-Stokes problem (2.10), we use the so-called $P_N P_N$ space, $(\boldsymbol{u}, p) \in (V_p, P_p)$ [61].

Note that, unlike the DG formulation, these matrices are not element-wise block diagonal. In fact, the support of these bases is local for only those i, j contained fully within one high-order element, and extends across two neighboring elements for those i, j on the element boundaries.

As a representative example, take the assembly of $\boldsymbol{L}$. Consider a single element K_k and its local basis functions $\boldsymbol{\phi}_i^k$, given by the restriction of the global basis functions $\boldsymbol{\phi}_i$ to this element. There is a global-to-local mapping $\mathcal{E}_k$ for element K_k that relates the global indices to the corresponding local basis indices. Thus, we can define a local matrix $\boldsymbol{L}^k$ by considering only the contribution from element K_k to the global $\boldsymbol{L}$ as

$$\boldsymbol{L}_{ij}^k = \int_{K_k} \boldsymbol{\phi}_i^k \boldsymbol{\phi}_j^k \; dx. \tag{3.12}$$

The global matrix is then assembled by adding up all the local contributions,

$$\boldsymbol{L} = \sum_{k=1}^{n_k} \mathcal{E}_k^T \boldsymbol{L}_{ij}^k. \tag{3.13}$$

In this manner, both the DG and FEM system matrices reduce to element-wise local operations instead.

3.2 Sum-factorization techniques

A traditional assembly of the local matrices would require evaluating all $n_c(p+1)^d \times n_c(p+1)^d$ entries by integrating the two functions across a set of $(p+1)^d$ quadrature nodes. Storing the matrices would therefore require $\mathcal{O}(p^{2d})$ memory. Moreover, this naive assembly typically costs $\mathcal{O}(p^{3d})$ operations. When running simulations at high polynomial degrees, these costs can be prohibitive. This is particularly true on GPUs, where the system matrices may be too large to fit in device memory. Therefore, we turn to sum-factorization techniques to replace our matrix-vector products with on-the-fly evaluations of the integrals [125, 100].

For the purposes of illustration, let us consider a matrix-free implementation of the local scalar stiffness matrix L on a hexahedral ($d = 3$) element K:

$$L_{ij}^K = \int_K \nu \nabla \phi_i \cdot \nabla \phi_j \; d\boldsymbol{x}. \tag{3.14}$$

We assume our isoparametric mapping $T^K : \mathcal{R} \to K$ has Jacobian J^K. Mapping to the reference cube, we have

$$L_{ij}^K = \int_{\mathcal{R}} \nu \nabla \phi_i^T \left(J^K\right)^{-T} \left|J^K\right| \left(J^K\right)^{-1} \nabla \phi_j \; d\boldsymbol{\xi}, \tag{3.15}$$

where now the ∇ operators and basis functions $\{\phi_i\}_{i=1}^{n_v}$ are understood to be in the reference space, parameterized by $\boldsymbol{\xi} \in \mathbb{R}^d$. Choosing our bases to be the tensor product of 1D Lagrange interpolating polynomials defined on the $p+1$ Legendre-Gauss-Lobatto points, we have

$$\phi_{ijk}(\boldsymbol{\xi}) = \phi_i(\xi_1)\phi_j(\xi_2)\phi_k(\xi_3), \tag{3.16}$$

for all $0 \leq i, j, k \leq p$. Choosing n_q^d tensor-product quadrature points to evaluate the integrals in (3.15), we can also denote the quadrature nodes and weights using this same multi-index notation. That is, the quadrature weights and points are $\left\{w_{i_q j_q k_q}\right\}_{i_q,j_q,k_q=1}^{n_q}$ and $\left\{\boldsymbol{\xi}_{i_q j_q k_q}\right\}_{i_q,j_q,k_q=1}^{n_q}$, respectively. Thus, (3.15) becomes

$$\begin{aligned} L_{ij}^K = \sum_{i_q,j_q,k_q=1}^{n_q} \nu w_{i_q j_q k_q} \nabla\phi_i^T \left(\boldsymbol{\xi}_{i_q j_q k_q}\right) \left(J^K(\boldsymbol{\xi}_{i_q j_q k_q})\right)^{-T} \\ \left|J^K(\boldsymbol{\xi}_{i_q j_q k_q})\right| \left(J^K(\boldsymbol{\xi}_{i_q j_q k_q})\right)^{-1} \nabla\phi_j \left(\boldsymbol{\xi}_{i_q j_q k_q}\right). \end{aligned} \tag{3.17}$$

As an aside, the solution u evaluated at a quadrature point is

$$u\left(\boldsymbol{\xi}_{i_q j_q k_q}\right) = \sum_{i,j,k=0}^{p} u_{ijk}\phi_{ijk}\left(\boldsymbol{\xi}_{i_q j_q k_q}\right) \tag{3.18}$$

$$= \sum_{k=0}^{p} \phi_k(\xi_{k_q}) \sum_{j=0}^{p} \phi_j(\xi_{j_q}) \sum_{i=0}^{p} u_{ijk}\phi_i(\xi_{i_q}). \tag{3.19}$$

We immediately notice that the number of operations required to evaluate a function at each of the quadrature points is $\mathcal{O}(p^4)$, or $\mathcal{O}(p^{d+1})$ in general.

Kronecker products allow us to simplify notation. First, we define the one-dimensional Gauss point evaluation matrix as the $n_q \times (p+1)$ Vandermonde-type matrix obtained by evaluating each of the one-dimensional basis functions at all of the quadrature points:

$$B_{i_q,j} = \phi_j(\xi_{i_q}). \tag{3.20}$$

With this notation, (3.19) is equivalent to the computation of the Kronecker product

$$u\left(\boldsymbol{\xi}\right) = \left(B \otimes B \otimes B\right) u. \tag{3.21}$$

Likewise, we define the 1D Gauss point differentiation matrix as

$$D_{i_q,j} = \frac{d\phi_j}{d\xi}\left(\xi_{i_q}\right). \tag{3.22}$$

Returning to (3.17), we are now ready to recast the operator as a Kronecker product of local 1D matrices. First, we precompute the tensor $W \in \mathbb{R}^{n_q^d \times n_q^d \times d \times d}$, which is diagonal in its first two dimensions and is defined by

$$W_{i_q j_q k_q, i_q j_q k_q} = \nu w_{i_q j_q k_q} \left(J^K(\boldsymbol{\xi}_{i_q j_q k_q})\right)^{-T} \left|J^K(\boldsymbol{\xi}_{i_q j_q k_q})\right| \left(J^K(\boldsymbol{\xi}_{i_q j_q k_q})\right)^{-1}. \tag{3.23}$$

Next, we recognize that evaluating $\nabla\Phi$ at all the quadrature points means contracting with the tensor $\boldsymbol{G}_\phi \in \mathbb{R}^{n_q^d \times (p+1)^d \times d}$ defined by

$$\boldsymbol{G}_\phi = \begin{bmatrix} B \otimes B \otimes D \\ B \otimes D \otimes B \\ D \otimes B \otimes B \end{bmatrix}. \tag{3.24}$$

Therefore, the local operator (3.17) is equivalent to

$$L^K = \boldsymbol{G}_\phi^T W \boldsymbol{G}_\phi, \tag{3.25}$$

where the transpose is taken only over the first two dimensions of $\boldsymbol{G}_\phi$. A major benefit of reformulating the local operator into this form is that it is no longer necessary to form a global matrix. Instead, the action of this operator can be recreated using the 1D matrices B and D and the precomputed tensor W. Precomputing W requires $\mathcal{O}(p^d)$ operations and $\mathcal{O}(p^d)$ memory. Applying the operator $\boldsymbol{G}_\phi$ and its transpose requires $\mathcal{O}(p^{d+1})$ operations and $\mathcal{O}(p^d)$ memory.

If W is computed by interpolating from the nodal points using sum-factorization techniques, then the setup costs would increase to $\mathcal{O}(p^{d+1})$. To achieve our leading-order costs, we assume that J (and therefore W) can be computed or is available at quadrature points in $\mathcal{O}(1)$ time. Indeed, our implementation stores J at quadrature points.

Using these sum-factorization techniques, we can also derive matrix-free implementations of $\boldsymbol{M}$, $\boldsymbol{G}$, D, and $\boldsymbol{N}(\boldsymbol{u})$ from their definitions. Each operator has comparable computational cost and memory requirements as in the case of this scalar diffusion operator.

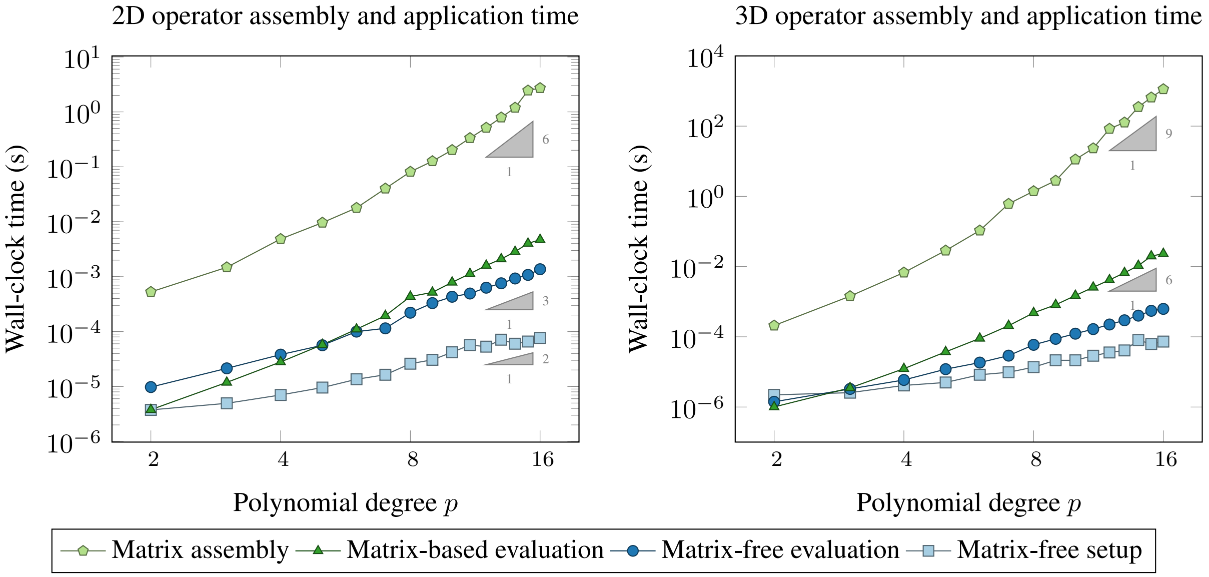


Figure 3.1: Run time comparison of standard matrix-based and sum-factorized matrix-free operator setup and evaluation for scalar Laplacian in 2D and 3D. Standard (naive) matrix assembly scales like $\mathcal{O}(p^{3d})$ and matrix-based operator evaluation scales like $\mathcal{O}(p^{2d})$, while matrix-free setup scales like $\mathcal{O}(p^d)$ and matrix-free operator evaluation scales like $\mathcal{O}(p^{d+1})$.

Figure 3.1 shows run times for sum-factorized matrix-free operator setup and evaluation compared with standard matrix assembly and matrix-based operator evaluation for polynomial degrees between 2 and 16. We notice that matrix assembly is the most expensive operation for all the cases tested, usually taking about two orders of magnitude more time than operator evaluation. In 2D, our implementation of matrix-free operator evaluation is more efficient than matrix-based operator evaluation for polynomial degrees greater than 5. In 3D, the matrix-free sum-factorized operator evaluation is more efficient for polynomial degrees greater than 2. In the matrix-free context, the setup and precomputations typically represent a negligible portion of the overall cost of using these operators.

3.3 Proposed matrix-free operator decomposition

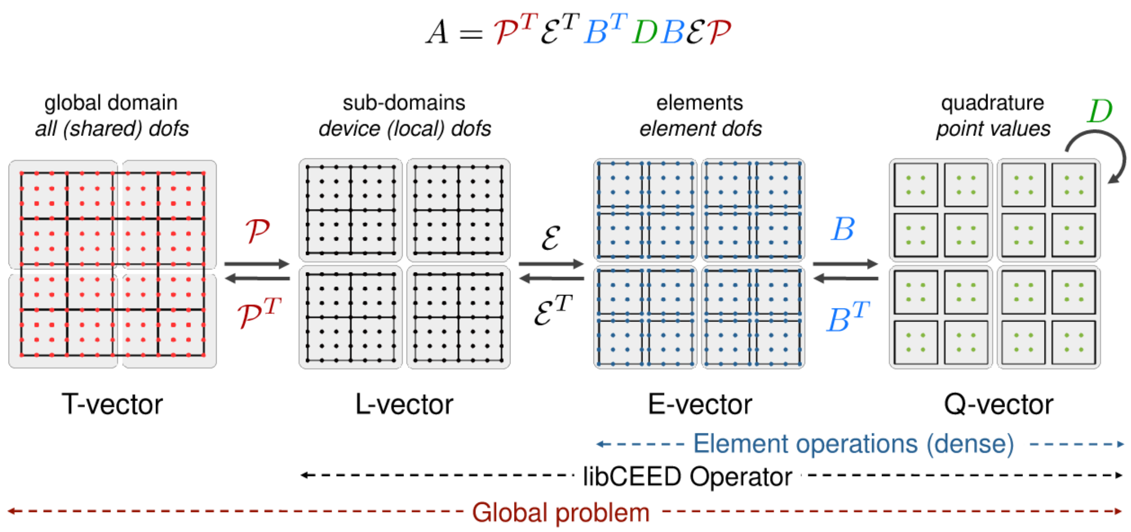


Figure 3.2: Decomposition of a high-order operator into efficient sub-operators.

Following the CEED framework [46], we now fully describe matrix-free operator evaluation for an arbitrary Galerkin operator, as shown in Figure 3.2. In particular, we write

$$y = Ax = \mathcal{P}^T \mathcal{E}^T \mathcal{B}^T D \mathcal{B} \mathcal{E} \mathcal{P} x, \tag{3.26}$$

for an operator that maps a vector of true DoFs in the trial space, x, to a vector of true DoFs in the test space, y. The operator first maps to local processors through the $\mathcal{P}$ restriction operator, then map to element-wise local operations through the $\mathcal{E}$ restriction operator, then map to quadrature points through the $\mathcal{B}$ operator. At each quadrature point, pointwise local operations are performed composing the underlying mathematics of the specific operator. Finally, the operator maps back to the test space true DoFs by applying the transpose of the previous sequence of restriction operators.

The only communication across processors occurs in the evaluation of the $\mathcal{P}$ and $\mathcal{P}^T$ operators. Since our implementation is designed for GPUs, on which communication overheads are expensive, this step should be avoided whenever possible to minimize communication. A performant implementation should therefore store the evaluation of the operator in device-local DoFs for instance when the program is run on only one GPU to avoid triggering the evaluation of $\mathcal{P}$ or $\mathcal{P}^T$. Moreover, these operators could be combined with communication-avoiding algorithms [65, 89] to minimize costly transfers to and from the device during higher-level algorithms.

As discussed previously, DG operators are naturally element-wise diagonal, so $\mathcal{E} = I$ for a general DG operator. On the other hand, FEM operators require the cheap reduction operation shown in (3.13). More advanced methods involving adaptive mesh refinement [32] would enhance this restriction operator with further interpolation, but that is beyond the scope of this thesis.

Section 3.2 fully explores how to efficiently implement the remaining element-wise local operations. In particular, sum-factorization reduces the cost of the libCEED $\mathcal{B}$ operator to $\mathcal{O}(p^{d+1})$ operations, as described by (3.24). In the case of different test and trial spaces, such as the discrete divergence operator (3.7), the specific operators mapping to and from the quadrature values are different. Regardless, we again utilize sum-factorization techniques to achieve the same leading-order costs, up to a constant factor.

3.4 GPU implementation

We have implemented the numerical algorithms described above in the framework of the MFEM finite element library [2, 86]. These algorithms take the form of single-source compute kernels that can target several different backends, including OpenMP on traditional CPUs as well as CUDA for use on the GPU. To utilize this functionality in MFEM yourself, simply set the “assembly level” of the bilinear form of most Galerkin operators to “partial.” In this section, we describe practical details that were required to obtain performant implementations for these algorithms.

As previously mentioned, minimizing memory movement between the CPU and GPU is crucial. Modern GPU architectures have limited memory and cache sizes compared to their CPU counterparts, but can reach higher peak performance in terms of floating point operations. This combination means that performance is only achievable if algorithms reach higher arithmetic intensities [78], thus motivating high-order matrix-free operators and solvers that have this potential.

We have found that using loop bounds known at compile time drastically improves performance. In p-dependent compute kernels, these compile-time constants permit shared memory access within an element, thus reducing memory allocations and movement. Figure 3.3 shows the benefits of utilizing shared memory for the intra-element operations. We see that the shared-memory implementation outperforms the naive implementation by a factor of 4-8. In

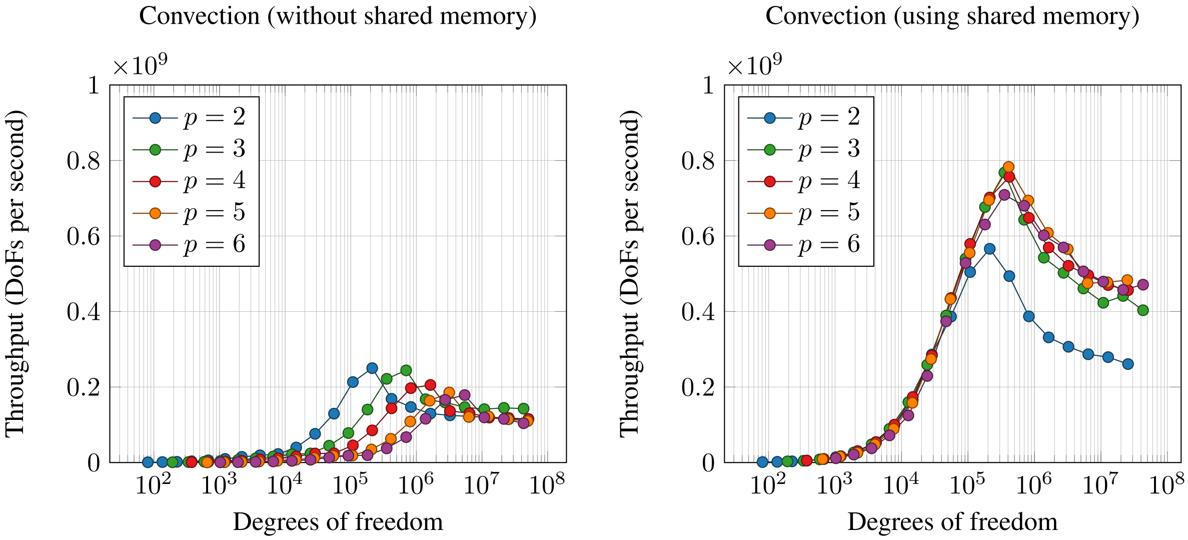


Figure 3.3: Throughput for nonlinear vector convection evaluation $\boldsymbol{N}(\boldsymbol{u})$ in 3D for several polynomial degrees on the GPU. Left: Initial implementation. Right: Element-wise shared-memory implementation. Efficiently reusing shared memory increases throughput by a factor of between 4 and 8 and also allows for the solution of larger problems.

practice, just-in-time compilation or explicit template instantiation can be used to allow for arbitrary-order simulations.

Figure 3.4 shows the throughput plots of our optimized implementations of the matrix-free linear operator evaluations in 3D. These results were performed on a single Nvidia V100 GPU, showing the throughput achieved for various problem sizes and polynomial degrees. Similar benchmark problems were used to assess kernel and backend performance in the context of the CEED project [46].

We believe that our implementation can be improved by better taking advantage of the GPU's shared memory. First of all, symmetries exist in the 1D interpolation and differentiation matrices (3.20) and (3.22) since the Legendre-Gauss-Lobatto nodes are symmetric about the origin. Taking advantage of these symmetries would approximately halve shared memory usage and simultaneously increase the arithmetic intensity of all operator evaluations. In some sense, this improvement can be viewed as an extension of the original sum-factorization techniques, which use inherent operator symmetries to avoid extra memory storage. Moreover, we see from Figure 3.4 that our implementation achieves lower throughput at lower orders. It may be possible to address this issue by combining several low-order elements per thread block to yield higher throughput. Unfortunately, the shared-memory requirements of our approach increase with order p and dimension d, so exhausting the shared memory is inevitable with very high order simulations. If necessary, one could compensate by storing

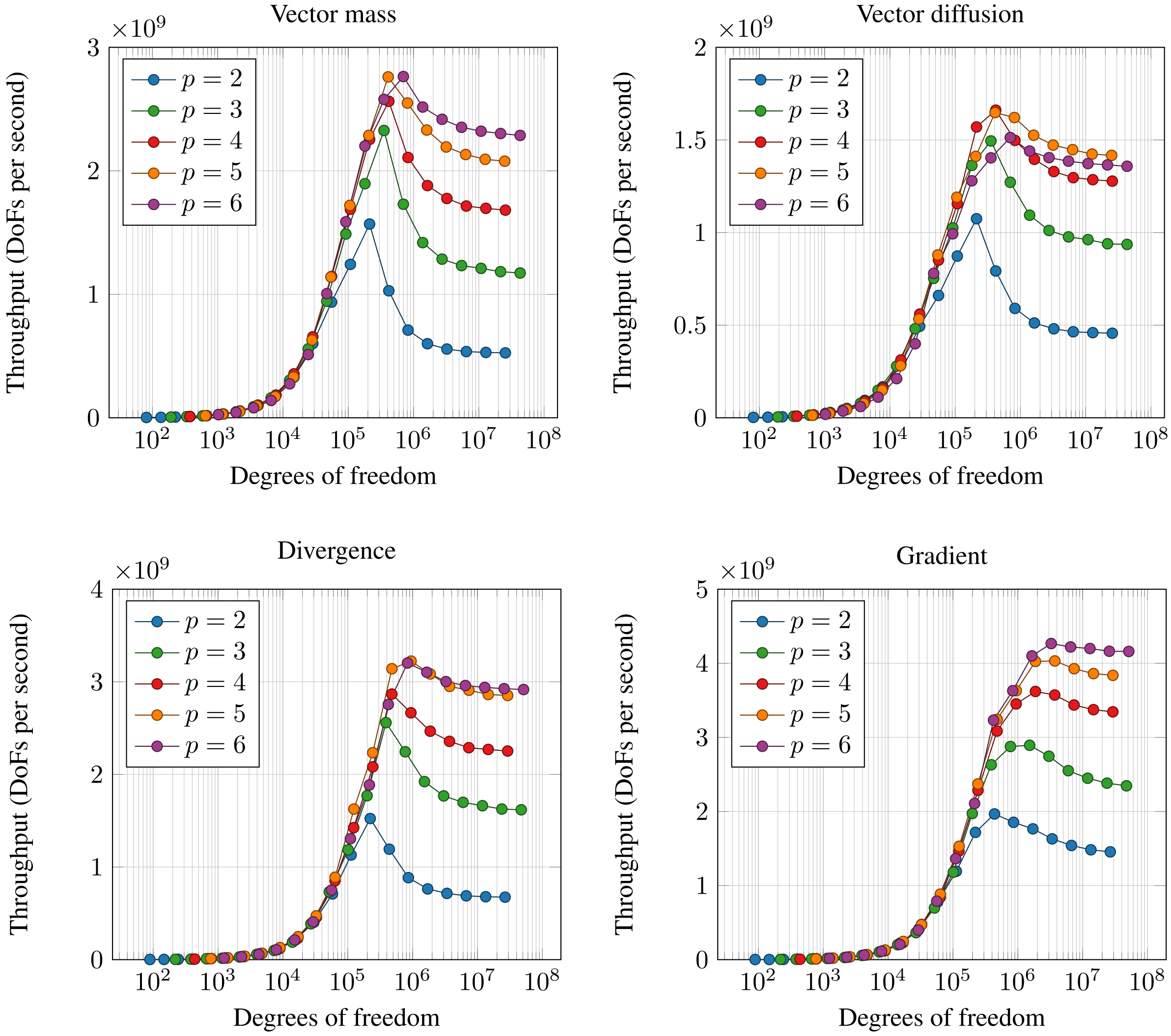


Figure 3.4: Throughput for linear operator evaluation kernels $\boldsymbol{M}$, $\boldsymbol{L}$, $\boldsymbol{D}$, G in 3D for orders $p = 2$ to 6 on the GPU. Maximum throughput is achieved for higher orders ($p > 3$) and larger problems (more than 10^6 DoFs).

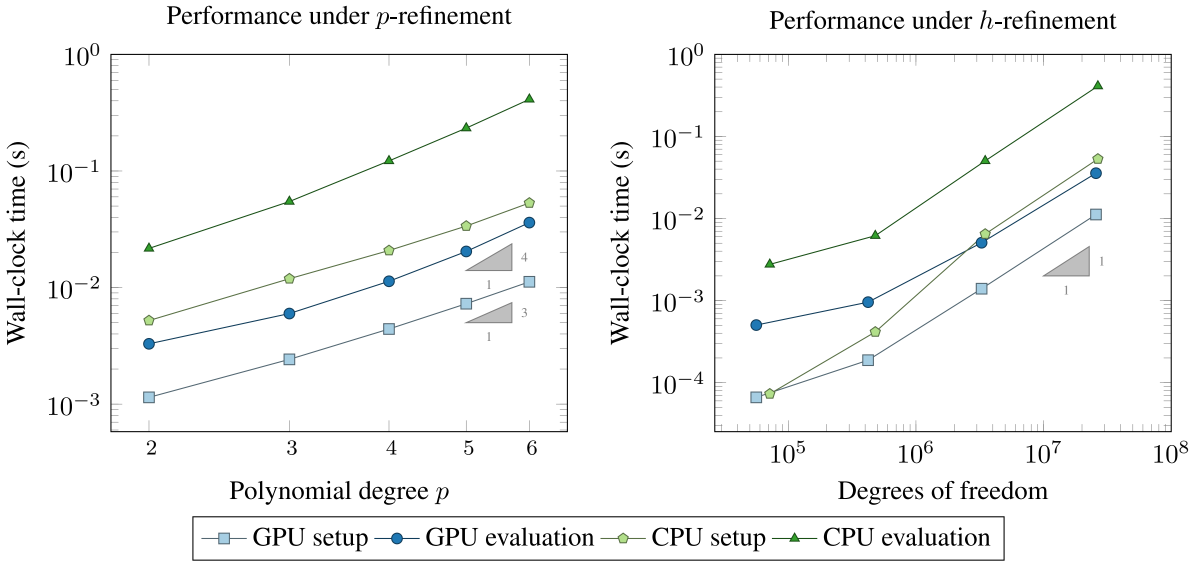


Figure 3.5: Performance of unsteady Stokes operator in 3D. Our implementation achieves expected rates of $\mathcal{O}(p^d)$ and $\mathcal{O}(p^{d+1})$ for matrix-free setup and evaluation of the block operator, respectively. Shared-memory GPU implementation of operator evaluation outperforms 20-core CPU implementation by a factor of 6 at $p = 2$ and a factor of 11 at $p = 6$.

B and D in the GPU's global memory to allow for even higher order simulations.

3.4.1 Unsteady Stokes flow

We next evaluate the performance of our GPU implementation of the block, unsteady Stokes operator. Figure 3.5 presents the 3D operator setup and evaluation times under p-refinement for a fixed mesh with 32,768 hexahedral elements. The GPU results were computed using one Nvidia V100 GPU, while the CPU results used 20 POWER8 cores on one node of Lawrence Livermore National Laboratory's Ray supercomputer. Our implementation achieves the expected rates of $\mathcal{O}(p^d)$ and $\mathcal{O}(p^{d+1})$ for matrix-free setup and evaluation, respectively. Moreover, we can see the computational benefits of high-order simulations on the GPU, as our optimized GPU kernels outperform the 20-core CPU implementation by a factor of 6 at $p = 2$ and a factor of 11 at $p = 6$.

Figure 3.5 also shows the performance under h-refinement, fixing the polynomial degree at $p = 6$. We see that for fixed p, the wall-clock time scales linearly with DoFs. Empirically, we see that this linear scaling occurs for problems with more than $\mathcal{O}(10^6)$ DoFs, while sublinear scaling occurs for problems with less than $\mathcal{O}(10^6)$ DoFs. This probably occurs because larger problems better saturate the GPU, thus attaining larger throughput. Once the device is saturated, the operator evaluation is accelerated by a factor of about 11 relative to the 20-core CPU evaluation.

3.5 Summary

In this chapter, we have described a high-order finite element method for incompressible flow problems with a matrix-free implementation that efficiently utilizes the high performance and memory bandwidth of modern graphics processing units. The resulting finite element operators used sum-factorization algorithms and optimized GPU kernels to obtain throughput of over a billion degrees of freedom per second on a single Nvidia V100 GPU. These same techniques were shown to immediately apply to discontinuous Galerkin operators as well. These and several other high-order Galerkin operators have been implemented in the open source MFEM library [86].

Chapter 4

Matrix-free preconditioning of incompressible flow with low-order refined preconditioners

Utilizing the matrix-free operators described in Chapter 3 allows us to compose the vector $\boldsymbol{b}$ and matrix-vector product $A\boldsymbol{x}$ from (2.53) efficiently. And indeed, for compressible flow problems where explicit time stepping can be employed, these are already the main challenges to overcome for matrix-free operators implemented on the GPU to be used effectively [73, 131]. However, most incompressible flow solvers require the solution of large, sparse linear systems [45], thus motivating the development of matrix-free solvers. Krylov subspace methods are a natural choice for matrix-free solvers, but they require effective preconditioners in order to obtain good performance [14]. Therefore, in this chapter we develop matrix-free preconditioners to solve the linear systems arising from high-order tensor-product finite element discretizations of the steady Stokes (2.13), unsteady Stokes (2.12), and unsteady incompressible Navier-Stokes equations (2.10). Particular emphasis is placed on solver robustness with respect to discretization and mesh parameters.

In recent years, there has been much work on the topic of matrix-free preconditioning for high-order discretizations. Matrix-free multigrid methods using point Jacobi and Chebyshev smoothing were considered in [112] and [78]. Matrix-free tensor-product approximations to block Jacobi preconditioners for discontinuous Galerkin discretizations were constructed in [100] and [101]. A number of other matrix-free methodologies for high-order discontinuous Galerkin flow solvers have been proposed, using techniques such as multigrid and block preconditioning [11, 56, 51]. In this chapter, we define sparse, low-order refined preconditioners [94, 44, 30] with parallel subspace corrections for diffusion problems [98]. Then we extend them to create a suite of incompressible flow preconditioners that are robust in both the mesh size and polynomial degree.

4.1 Revisiting temporal discretization

As opposed to the time-dependent conservation laws for which implicit methods ultimately yield a backward Euler-type system (2.53), incompressible flow is complicated by the fact that there is no temporal evolution equation corresponding to the pressure. Indeed, after spatial discretization of the time-dependent problems (2.10) and (2.12), we obtain (3.9) and (3.3), which are systems of differential-algebraic equations (DAEs) [69, 107]. We discretize these DAEs in time by split (i.e. projection or fractional step) or unsplit methods. For the unsplit methods, we use the method of lines to first discretize in space and then temporally discretize the resulting system of ordinary differential equations (3.3) and (3.9). Here, we again use DIRK schemes as our time-integration method [1]. On the other hand, split methods such as projection-type methods can be developed in order to decouple the solution of the velocity and pressure components. As a result, these methods can be computationally efficient, at the cost of incurring splitting and other approximation errors. Each of these methods is discussed in greater detail in the following sections.

4.1.1 DIRK schemes for DAEs

First, we reformulate the DIRK method in a manner suitable for solving DAEs [62, 19]. For this reformulation, we require that the DIRK scheme be stiffly accurate, i.e. that $a_{si} = b_i$ and $c_s = 1$. Then, considering the general differential-algebraic system

$$\begin{aligned} M\dot{\boldsymbol{y}} &= \boldsymbol{r}(t, \boldsymbol{y}, \boldsymbol{z}), \\ 0 &= \boldsymbol{g}(t, \boldsymbol{y}), \end{aligned} \tag{4.1}$$

we define the approximate solution to the differential variable $\boldsymbol{y}$ at the i^{th} stage by

$$\boldsymbol{y}_i^n = \boldsymbol{y}^n + \Delta t \sum_{j=1}^{i} a_{ij} \boldsymbol{k}_j^n. \tag{4.2}$$

Analogously to (2.47), the stage derivatives $\boldsymbol{k}_i^n$ are given by

$$M\boldsymbol{k}_i^n = \boldsymbol{r}(t^n + \Delta t c_i, \boldsymbol{y}_i^n, \boldsymbol{z}_i^n), \tag{4.3}$$

Multiplying (4.2) by the mass matrix M and inserting (4.3), we obtain

$$M\boldsymbol{y}_i^n = M\boldsymbol{y}^n + \Delta t \sum_{j=1}^{i} a_{ij} \boldsymbol{r}(t^n + \Delta t c_j, \boldsymbol{y}_j^n, \boldsymbol{z}_j^n), \tag{4.4}$$

which, when augmented with the constraint

$$0 = \boldsymbol{g}(t, \boldsymbol{y}_i), \tag{4.5}$$

results in a system of equations for the i^{th} stage approximations for the differential and algebraic variables $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$. Because the DIRK schemes under consideration are stiffly accurate, the values at the next time step are given by the final stage approximations,

$$\boldsymbol{y}^{n+1} = \boldsymbol{y}_s^n, \qquad \boldsymbol{z}^{n+1} = \boldsymbol{z}_s^n. \tag{4.6}$$

Just as in Section 2.3, we apply Newton's method to this potentially nonlinear system of equations to arrive at a set of linear systems we must solve efficiently. Applying this method to the semi-discrete Stokes problem (3.3) ultimately requires the solution to the linear system,

$$\begin{bmatrix} \frac{1}{\alpha\Delta t}\boldsymbol{M} + \boldsymbol{L} & \boldsymbol{G} \\ -D & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_i^n \\ p_i^n \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}_i \\ 0 \end{bmatrix}, \tag{4.7}$$

every Runge-Kutta stage, with the right-hand side $\boldsymbol{F}_i$ given by

$$\boldsymbol{F}_i = \frac{\boldsymbol{M}\boldsymbol{u}^n}{\alpha\Delta t} + \boldsymbol{f}_i + \frac{1}{\alpha}\sum_{j=1}^{i-1} a_{ij}(\boldsymbol{f}_j - \boldsymbol{L}\boldsymbol{u}_j^n - Gp_j^n). \tag{4.8}$$

This fully discrete linear system (4.7) and its steady state counterpart (3.8) are saddle point problems [14]. We present robust matrix-free solvers for such saddle point systems in Section 4.2.4.

4.1.2 Projection methods

Projection methods, first introduced by Chorin in 1967, are a class of split methods for the temporal integration of the incompressible Navier-Stokes equations [33, 34]. These methods have the attractive feature that they only require the solution to uncoupled, positive-definite problems, instead of the coupled, saddle-point type problems that are required by the DIRK schemes described above. For this reason, projection and fractional-step methods have become extensively used for incompressible flow problems [45, 106]. Chorin's original method has since been modified and extended to a wide range of variants [12, 24, 71, 127, 95]. See [61] for a full review and analysis of a selection of these variants.

Following the method presented in [127], we use equal order polynomial degrees for velocity and pressure, often known as a $P_N P_N$ formulation. This method uses an implicit-explicit time-integration scheme for the viscous and convective terms respectively, thereby avoiding the need to solve a nonlinear system of equations at every time step. We use a backward differentiation formula (BDF) of order k for the implicit terms and an extrapolation method of order k for the explicit terms with corresponding coefficients b_j and a_j [127, 95, 62]. First, we introduce the linear term $L(\boldsymbol{u}) = \nu\Delta\boldsymbol{u}$ and nonlinear term $N(\boldsymbol{u}) = -(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}$ as well as their time-extrapolated versions,

$$L^*(\boldsymbol{u}^{n+1}) = \sum_{j=1}^{k} a_j L(\boldsymbol{u}^{n+1-j}), \tag{4.9}$$

$$N^*(\boldsymbol{u}^{n+1}) = \sum_{j=1}^{k} a_j N(\boldsymbol{u}^{n+1-j}). \tag{4.10}$$

Directly applying a k-step BDF method to (2.10) yields

$$\sum_{j=0}^{k} \frac{b_j}{\Delta t}\boldsymbol{u}^{n+1-j} = -\nabla p^{n+1} + L(\boldsymbol{u}^{n+1}) + N^*(\boldsymbol{u}^{n+1}) + \boldsymbol{f}^{n+1}, \tag{4.11}$$

where $\boldsymbol{f}^{n+1}$ is assumed to be known *a priori*. Introducing $\boldsymbol{F}^*(\boldsymbol{u}^n)$ to represent all known terms at a given time step,

$$\boldsymbol{F}^*(\boldsymbol{u}^n) = -\sum_{j=1}^{k} \frac{b_j}{\Delta t}\boldsymbol{u}^{n+1-j} + N^*(\boldsymbol{u}^{n+1}) + \boldsymbol{f}^{n+1}, \tag{4.12}$$

we can simplify (4.11) to

$$\frac{b_0}{\Delta t}\boldsymbol{u}^{n+1} = -\nabla p^{n+1} + L(\boldsymbol{u}^{n+1}) + \boldsymbol{F}^*(\boldsymbol{u}^n). \tag{4.13}$$

Unfortunately, despite using a $k > 1$ order time-integration scheme, this method yields at most first-order convergence in time for velocity, as shown in [71] and later proved by [61]. This is caused by splitting errors and large divergence errors on the boundary of the domain. Therefore we use the velocity-correction formulation presented in [71], where the linear term $L(\boldsymbol{u})$ is instead expressed as

$$L_\times(\boldsymbol{u}) = \nu\nabla(\nabla\cdot\boldsymbol{u}) - \nu\nabla\times\nabla\times\boldsymbol{u}, \tag{4.14}$$

using well-known vector calculus identities. This alternative form of the linear term imposes the incompressibility constraint from (2.10) weakly, by setting the first term in (4.14) equal to zero.

In order to solve for pressure, we first rearrange (4.13) and take the divergence of both sides in order to get

$$\nabla p^{n+1} = -\frac{b_0}{\Delta t}\boldsymbol{u}^{n+1} + \underbrace{L^*_\times(\boldsymbol{u}^{n+1}) + F^*(\boldsymbol{u}^{n+1})}_{\tilde{F}^*(\boldsymbol{u}^{n+1})}, \tag{4.15}$$

$$\implies \Delta p^{n+1} = \nabla\cdot\tilde{F}^*(\boldsymbol{u}^{n+1}). \tag{4.16}$$

Notice how the first right-hand side term of (4.15) vanishes due to the incompressibility constraint. Equation 4.16 is closed by the boundary condition,

$$\nabla p^{n+1} \cdot \boldsymbol{n} = -\frac{b_0}{\Delta t}\boldsymbol{u}^{n+1} \cdot \boldsymbol{n} + \tilde{F}^*(\boldsymbol{u}^{n+1}) \cdot \boldsymbol{n} \quad \text{on } \partial\Omega, \tag{4.17}$$

where $\boldsymbol{n}$ is the outward pointing normal vector. We use the known Dirichlet boundary condition $\boldsymbol{u}^{n+1} \cdot \boldsymbol{n} = \boldsymbol{g}_D^{n+1} \cdot \boldsymbol{n}$ to evaluate (4.17). In the case of a pure Neumann boundary condition, we close the system with a mean-zero condition on pressure:

$$\int_\Omega p \, dx = 0. \tag{4.18}$$

Therefore, this projection method computes $\boldsymbol{u}^{n+1}$ in three steps. First, the extrapolated contributions from the nonlinear and forcing terms are combined to compute $\boldsymbol{F}^*(\boldsymbol{u}^n)$ via (4.12). Second, we solve for p^{n+1} in the pressure-Poisson problem (4.16), closed with (4.17) and (4.18). Finally, we return to (4.13) and solve for $\boldsymbol{u}^{n+1}$ in the following Helmholtz problem:

$$\frac{b_0}{\Delta t}\boldsymbol{u}^{n+1} - L(\boldsymbol{u}^{n+1}) = -\nabla p^{n+1} + \boldsymbol{F}^*(\boldsymbol{u}^{n+1}) \qquad \text{in } \Omega, \tag{4.19}$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{g}_D^{n+1} \qquad \text{on } \partial\Omega. \tag{4.20}$$

This projection method is kth order in time for velocity (up to $k = 3$) [61]. As previously mentioned, a major benefit of this method is its computational efficiency. Each time step requires only one new nonlinear evaluation, one Poisson solve, and one Helmholtz solve. We will discuss the matrix-free solvers that we use for these sub-problems in Section 4.2.3.

4.2 Matrix-free preconditioners

The numerical methods described above require solving large, sparse linear systems. The fully discrete, steady Stokes equation requires solving the saddle-point linear system (3.8). The time discretization of the unsteady Stokes equation (2.12) by a DIRK method results in a sequence of saddle point problems (4.7). The velocity-correction schemes require the solution of a Poisson problem (4.16) for the pressure and a Helmholtz equation (4.19) for the velocity. Additionally, the nonlinear extrapolation requires the inversion of the velocity mass matrix.

The main challenge associated with the matrix-free solution of high-order flow problems is constructing efficient preconditioners that result in iteration counts that are independent of the discretization parameters h, p, and Δt. In this section, we describe the construction of robust preconditioners that do not require the assembly of the high-order system matrices. We begin by describing our preconditioning strategy for the relatively simpler sub-problems, which can then be combined to create effective preconditioners for the more challenging, coupled problems.

4.2.1 Collocated mass preconditioning

In order to precondition the mass matrix, we make use of a diagonal preconditioner based on collocated quadrature [49, 58]. Note that for the one-dimensional mass matrix,

$$M_{ij} = \int_{\mathcal{R}} \phi_i \phi_j \, dx, \tag{4.21}$$

the integrand is a polynomial of degree $2p+2$. Therefore, mapping from the nodes to the $(p+1)$-point Gaussian quadrature points, one can exactly integrate this expression and evaluate the mass entries (and its inverse). However, the exact mass matrix and its inverse in general are full, so inverting this matrix is not practical. Because we use a nodal Legendre-Gauss-Lobatto (LGL) basis for the finite element spaces, we instead seek to integrate at this same set of quadrature points. The immediate benefit of this decision is that the resulting mass matrix $\tilde{M}$ is by construction diagonal:

$$M_{ij} = \sum_{k=1}^{p+1} w_k \phi_i(\xi_k)\phi_j(\xi_k) \approx \sum_{k=1}^{p+1} w_k \delta_{ik}\delta_{jk} = w_i \delta_{ij} = \tilde{M}_{ij}. \tag{4.22}$$

This diagonal matrix is efficiently constructed and inverted in constant time and memory per degree of freedom. However, integrating an integrand of degree $2p+2$ with a rule that exactly integrates polynomials up to degree $2p+1$ introduces an under-integration error. Conveniently, this error converges away spectrally fast for smooth problems as we increase p [126]. Therefore, $\tilde{M}$ is spectrally equivalent to the fully-integrated mass matrix, with constants of equivalence independent of h and p. Thus, the number of $\tilde{M}$-preconditioned solver iterations remains uniformly bounded with respect to the mesh size and polynomial degree.

4.2.2 Low-order refined preconditioners for Poisson and Helmholtz problems

Matrix-free preconditioners for the symmetric positive-definite Poisson and Helmholtz problems form the fundamental building blocks for our robust fluid solvers. These preconditioners are described in detail in [98], and are based on the spectral equivalence between the high-order finite element discretization, and a low-order ($p_{low} = 1$) finite element discretization on an LGL-refined mesh. This equivalence is often referred to as the finite element method–spectral element method (FEM-SEM) equivalence [31]. The low-order finite element discretization results in a sparse matrix with $\mathcal{O}(1)$ nonzeros per row independent of p, the polynomial degree of the original high-order discretization. Therefore, the memory requirements and computational cost to assemble the low-order matrix are both optimal, scaling linearly in the number of degrees of freedom.

Consider a scalar Poisson or Helmholtz problem

$$A\boldsymbol{u} = \boldsymbol{b}, \tag{4.23}$$

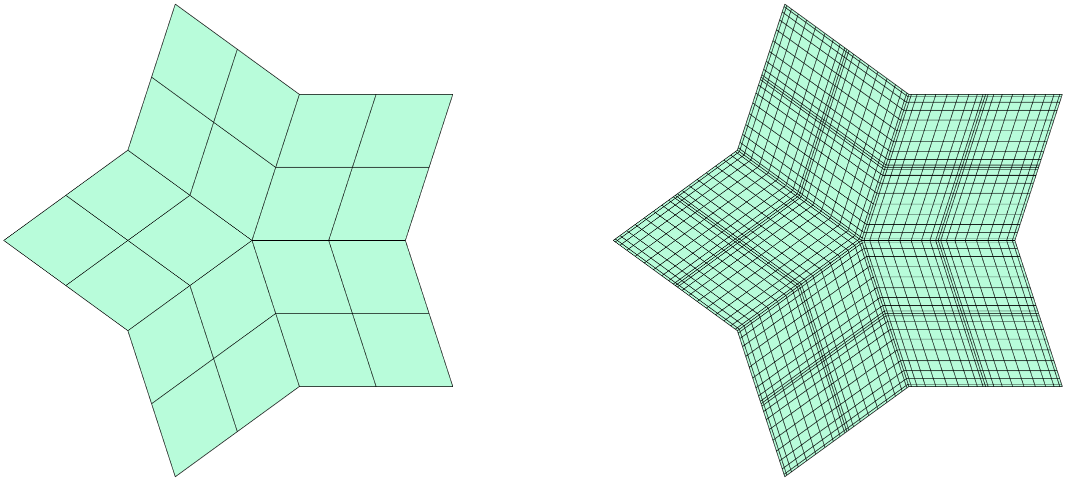


Figure 4.1: Illustration of the low-order refined methodology with $p = 10$, showing high aspect ratio elements near the coarse element interfaces. Left: original high-order mesh $\mathcal{T}_h$. Right: Legendre-Gauss-Lobatto refined mesh $\mathcal{T}_{LOR}$.

for $A = cM + L$, where c is a non-negative (but possibly zero) constant. We begin by constructing a low-order refined (LOR) operator,

$$A_{LOR} = cM_{LOR} + L_{LOR}. \tag{4.24}$$

The LOR operators are obtained by a standard piecewise linear finite element discretization on a refined mesh $\mathcal{T}_{LOR}$. This mesh is obtained by subdividing each element $K_i \in \mathcal{T}_h$ into the parallelepipeds defined by the Cartesian product of the $p + 1$ one-dimensional LGL points. Figure 4.1 illustrates one such low-order refined mesh. That is, one follows the same discretization presented in Section 3.1 for mesh $\mathcal{T}_{LOR}$, but sets $p_{LOR} = 1$. We use the identity operator to map DoFs from the high-order finite element LGL space to the low-order refined space. It can be shown that the low-order refined mass matrices and stiffness matrices M_{LOR} and L_{LOR} are spectrally equivalent to their high-order counterparts, M and L [28, 31, 29]. This equivalence directly follows from the one dimensional equivalences in the L^2 norm and H^1 seminorm:

Proposition 4.2.1. *There exists constants c and c' independent of p such that*

$$\frac{1}{c} \|\phi_p\|_{L^2([0,1])} \leq \|\phi_{LOR}\|_{L^2([0,1])} \leq c \|\phi_p\|_{L^2([0,1])}, \tag{4.25}$$

$$\frac{1}{c'} \|\phi_p'\|_{L^2([0,1])} \leq \|\phi_{LOR}'\|_{L^2([0,1])} \leq c' \|\phi_p'\|_{L^2([0,1])}. \tag{4.26}$$

Therefore, A_{LOR} is spectrally-equivalent to A, and a robust preconditioner for A_{LOR} is, in turn, a robust preconditioner for the original high-order matrix A. The advantage of the matrix A_{LOR} over A is its greatly increased sparsity, requiring only $\mathcal{O}(1)$ nonzeros per row. As a consequence, this matrix can be explicitly assembled and stored, as opposed to the original operator A implemented in a matrix-free manner. Having access to the actual entries of A_{LOR} allows for the construction of sophisticated preconditioners.

Note that each coarse element $K_i \in \mathcal{T}_h$ can be alternatively decomposed into a simplicial sub-mesh using the LGL points. It has been shown that preconditioners resulting from finite element discretizations using simplicial decompositions can result in improved convergence when compared with the tensor-product decomposition used in this chapter [54, 29]. Furthermore, several new configurations defining the low-order mesh were considered in [13], potentially improving convergence rates as well. However, the tensor-product decomposition we have presented has the advantage that the discretization primitives can be reused across both the high-order and low-order methods, greatly simplifying the implementation.

The main challenge associated with constructing effective preconditioners for A_{LOR} is the high aspect ratio associated with the low-order refined mesh $\mathcal{T}_{LOR}$ [83]. Because the LGL points are clustered near the endpoints of the interval, the resulting Cartesian product mesh consists of parallelepipeds with aspect ratios that scale like p [23]. As a result, the mesh $\mathcal{T}_{LOR}$ is not shape-regular with respect to p, and standard multigrid-type methods will not result in uniform convergence under p-refinement.

In order to address this issue, we make use of a structured geometric multigrid V-cycle with ordered ILU smoothing to treat the anisotropy of the problem, within a broader additive Schwarz framework. The additive Schwarz domain decomposition framework [128, 97] first decomposes the finite element space V_h into a sum of subspaces,

$$V_h = V_0 + \sum_{j=1}^{J} V_j. \tag{4.27}$$

On each subspace V_j, we let A_j be the restriction of A_h to V_j. Then the subspace operator $A_j : V_j \to V_j$, elliptic projection $P_j : V_h \to V_j$, and L^2 projection $Q_j : V_h \to V_j$ satisfy the useful identity

$$A_j P_j = Q_j A_h, \tag{4.28}$$

and so $P_j = A_j^{-1} Q_j A_h$. While inverting the subspace operator exactly is often not possible, an approximate local solver $R_j^{-1} \approx A_j^{-1}$ allows us to define the additive Schwarz preconditioner B^{-1} by

$$B^{-1} = \sum_{j=0}^{J} R_j^{-1} Q_j. \tag{4.29}$$

Each of the local approximate solvers R_j^{-1} may be applied in parallel. Thus, this method is often referred to as the method of parallel subspace corrections [136].

For our problem, we choose V_0 to be the coarse, low-order subspace defined on our original mesh $\mathcal{T}_h$,

$$V_0 = \left\{ v \in H^1(\Omega) \mid v(K_i) \in \mathcal{Q}^1(K_i) \ \forall K_i \in \mathcal{T}_h \right\}. \tag{4.30}$$

For the high-order methods we have developed on $\mathcal{T}_h$, the dimension of V_0 is much smaller than V_h, and indeed is independent of the polynomial degree p. Regardless, we are unable to use direct solvers on this space, so we choose the coarse solver R_0^{-1} to be one V-cycle of the algebraic multigrid method BoomerAMG [137] implemented in [50].

The spaces $V_1, \ldots, V_J$ are defined in terms of overlapping, unstructured patches of vertices from the original mesh $\mathcal{T}_h$. The vertices are partitioned into J disjoint sets $E_1, \ldots, E_J$. We associate a subdomain $\Omega_j \subseteq \Omega$ obtained by taking the union of all coarse elements $K \in \mathcal{T}_h$ containing any vertex $x_k \in E_j$. Thus, the subdomains overlap by a layer of elements only one element thick, although multiple subdomains may share the same overlap region.

For these local patches, we define R_j^{-1} to be an element-structured geometric multigrid V-cycle [22] as follows. Each level of the hierarchy is constructed from the previous level by removing half of the interior LGL points within each element, resulting in $\mathcal{O}(\log p)$ levels. This can be accomplished for instance by deleting every other interior one-dimensional LGL point within the elements composing Ω_j. Thus, the last level corresponds with the base $p = 1$ elements from $\mathcal{T}_h$. Importantly, we only use the low-order refined operators defined on these levels, ensuring operator memory and costs are independent of p.

At each level of the multigrid hierarchy, we perform one presmoothing and one postsmoothing step based on incomplete LU (ILU) smoothing. Since the effectiveness of an ILU solver strongly depends upon the ordering of the degrees of freedom, we choose our ordering based on the minimum discarded fill algorithm applied to unstructured meshes [43, 105]. This ordering ensures the number of multigrid iterations is independent of h and p [98]. Taking advantage of the additive Schwarz framework, the multigrid algorithm with ILU smoothing described above is applied in parallel to each of the subdomains independently, leading to an efficient and robust preconditioner.

4.2.2.1 Relationship to other Schwarz-based solvers

A number of other matrix-free solvers based on a Schwarz methodology have been proposed for the solution of the high-order Poisson problem, and by extension the incompressible Navier-Stokes equations. Closely related to the methods presented in this chapter, multigrid solvers with matrix-free Schwarz-based smoothers for the spectral element method were constructed in [83]. These methods were later extended in [55] to solve the unsteady Navier-Stokes equations, and were shown to perform efficiently on several large-scale tensor-product meshes. The additive Schwarz smoothers used in [83] and [55] are constructed using tensor-product subdomains corresponding to the spectral elements of the high-order discretization. Because each of the subdomains possesses a tensor-product geometry, the fast diagonalization method may be used to efficiently solve the local problems [135, 119].

In this chapter, however, we make use of subdomains defined by fully unstructured vertex patches, which in general do not possess a tensor-product structure. The application of fast diagonalization methods to such geometries is non-trivial [100]. For this reason, instead of using fast diagonalization to solve the local problems, we opt to solve the local low-order refined problems using the element-structured geometric multigrid V-cycle with ILU smoothing defined above.

4.2.3 Application to the projection method

The projection method described in Section 4.1.2 requires solvers for the vector mass matrix, the pressure Poisson problem (4.16) and the Helmholtz problem (4.19). Each of these problems is symmetric positive-definite, and so we can use a preconditioned conjugate gradient solver with the preconditioners described in Section 4.2.1 and Section 4.2.2. Because the pressure Poisson problem (4.16) with pure Neumann conditions has a null space consisting of constant functions, care must be taken to ensure that the right-hand side is orthogonal to the null space. Therefore, each application of the preconditioner is augmented with an orthogonalization step to ensure convergence.

4.2.4 Block preconditioners for Stokes

In order to solve the coupled Stokes problems (3.8) and (4.7), we make use of block-preconditioning techniques, allowing us to reuse the preconditioners for the Poisson and Helmholtz problems. Consider the representative saddle point system,

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{G} \\ D & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ p \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ 0 \end{bmatrix}. \tag{4.31}$$

We define the block triangular preconditioner

$$P_t = \begin{bmatrix} \boldsymbol{A} & 0 \\ D & S \end{bmatrix}, \tag{4.32}$$

where $S = -D\boldsymbol{A}^{-1}\boldsymbol{G}$ is the Schur complement. If we can afford the exact inversion of P_t^{-1}, then it can be shown that the spectrum of the preconditioned system $P_t^{-1}\boldsymbol{A}$ consists only of a single eigenvalue, and hence GMRES will converge in at most two iterations [14]. Applying this preconditioner requires solving linear systems with the matrices $\boldsymbol{A}$ and S. However, forming and inverting these systems is impractical, so in this section we provide matrix-free operators to approximate the action of $\boldsymbol{A}^{-1}$ and S^{-1}.

We replace the action of $\boldsymbol{A}^{-1}$ with the application of several uniformly-preconditioned conjugate gradient iterations on $\boldsymbol{A}$. For the steady Stokes system, $\boldsymbol{A} = \boldsymbol{L}$, so we can use the matrix-free low-order refined preconditioner defined in Section 4.2.2. Likewise, for the unsteady Stokes system, $\boldsymbol{A} = \frac{1}{\alpha\Delta t}\boldsymbol{M} + \boldsymbol{L}$, so we can use the previously-defined Helmholtz preconditioner. While these are vector linear systems, both $\boldsymbol{A}$ operators decouple the dimensions. Therefore, applying the scalar preconditioners d times (once in each dimension) directly provides a preconditioner for these vector linear systems. Since the CG iterations do not correspond to a fixed linear operator, it is important that we make use of flexible GMRES as the outer iterative method to solve the preconditioned system [114]. This subproblem need not be solved exactly, and empirically two or three CG iterations are sufficient to provide an effective preconditioner.

The Schur complement S is in general dense, and so now we focus on creating $\tilde{S}^{-1}$ to approximate the action of S^{-1}. For the steady Stokes system, $S = -D\boldsymbol{L}^{-1}\boldsymbol{G}$. Before creating an approximate solver for S, we notice that $L = -\nu D\boldsymbol{M}^{-1}\boldsymbol{G}$ by construction. To construct the approximate solver $\tilde{S}^{-1}$, we make use of the standard commutativity approximation [14]:

$$\boldsymbol{L}\boldsymbol{M}^{-1}\boldsymbol{G} \approx \boldsymbol{G}M^{-1}L. \tag{4.33}$$

Note that this approximation is, in fact, exact when the operators $\boldsymbol{L}$ and $\boldsymbol{G}$ commute, such as in the case of periodic boundary conditions. From (4.33), we have

$$\boldsymbol{M}^{-1}\boldsymbol{G}L^{-1}M \approx \boldsymbol{L}^{-1}\boldsymbol{G}, \tag{4.34}$$

and so

$$S = -D\boldsymbol{L}^{-1}\boldsymbol{G} \approx -D\boldsymbol{M}^{-1}\boldsymbol{G}L^{-1}M = \frac{1}{\nu}LL^{-1}M = \frac{1}{\nu}M. \tag{4.35}$$

That is, the mass matrix M provides an approximation to S. Therefore, we define the action of the approximate solver $\tilde{S}^{-1}$ by the diagonal mass preconditioner described in Section 4.2.1. In practice, approximating the action of S^{-1} by the action of $\tilde{S}^{-1}$ doubles to triples the iterations of the iterative solver. However, inverting S is infeasible, whereas applying $\tilde{S}^{-1}$ is efficiently performed in $\mathcal{O}(p^d)$ time.

Likewise, for the unsteady Stokes system, the Schur complement is given by

$$S = -D\left(\frac{1}{\alpha\Delta t}\boldsymbol{M} + \boldsymbol{L}\right)^{-1}\boldsymbol{G}. \tag{4.36}$$

Using the same commutativity approximation (4.33), we obtain

$$\boldsymbol{M}^{-1}\boldsymbol{G}\left(\frac{1}{\alpha\Delta t}M + L\right)^{-1}M \approx \left(\frac{1}{\alpha\Delta t}\boldsymbol{M} + \boldsymbol{L}\right)^{-1}\boldsymbol{G}. \tag{4.37}$$

Thus,

$$S \approx -D\boldsymbol{M}^{-1}\boldsymbol{G}\left(\frac{1}{\alpha\Delta t}M + L\right)^{-1}M = \frac{1}{\nu}L\left(\frac{1}{\alpha\Delta t}M + L\right)^{-1}M. \tag{4.38}$$

Therefore, for the unsteady Stokes system, we define

$$\tilde{S}^{-1} = \frac{\nu}{\alpha\Delta t}\tilde{L}^{-1} + \nu\tilde{M}^{-1}. \tag{4.39}$$

From (4.39), we can apply the action of $\tilde{S}^{-1}$ in a matrix-free manner by once again reusing the Poisson solver from Section 4.2.2 and the diagonal $\tilde{M}^{-1}$ from Section 4.2.1.

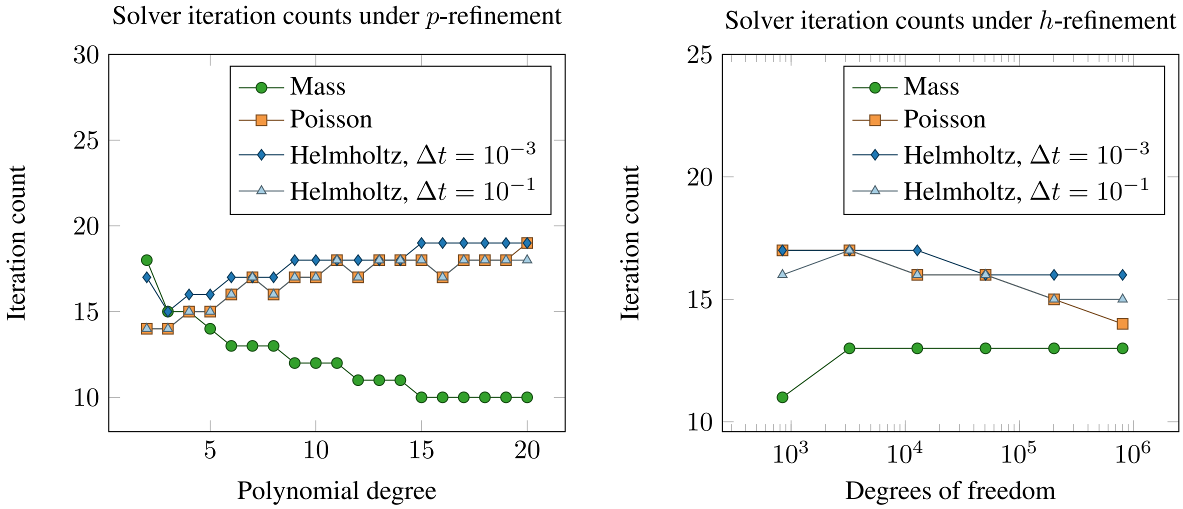


Figure 4.2: Iteration counts for sub-problem solvers under p- and h-refinement. For the case of h-refinement, we use a fixed polynomial degree of $p = 7$.

4.3 Numerical results

4.3.1 Sub-problem solver performance

We first assess the performance of the matrix-free sub-problem preconditioners by measuring the number of Krylov iterations required to converge to a fixed tolerance under h- and p-refinement. We solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$, where A is either the mass matrix, Laplacian operator, or positive-definite Helmholtz operator. We write the Helmholtz operator as $M/\Delta t + L$, and choose two representative time steps: $\Delta t = 10^{-1}$ and $\Delta t = 10^{-3}$. For each of the problems considered, we use a preconditioned conjugate gradient iteration to solve the problem, with a relative residual tolerance of 10^{-8} as a stopping criterion. The right hand side $\boldsymbol{b}$ is taken to be a random vector. For the mass matrix, the preconditioner is the collocated diagonal preconditioner described in Section 4.2.1, and for the Helmholtz and Poisson solvers, we use the low-order refined parallel subspace correction procedure described in Section 4.2.2.

To perform the p-refinement study, we use a fixed Cartesian grid with 64 elements in two dimensions, and polynomial degrees from $p = 2$ to $p = 20$. The number of iterations required to converge to tolerance is shown in Figure 4.2. We note that the number of iterations remains bounded for all problems and for all polynomial degrees. We observe a slight pre-asymptotic increase in the number of iterations for the Poisson and Helmholtz problems, but the iteration counts remain below 20 for all cases. As expected, the number of iterations required for the mass solve decreases with increasing polynomial degree. This result corroborates eigenvalue analysis of the preconditioned system, which shows decreasing

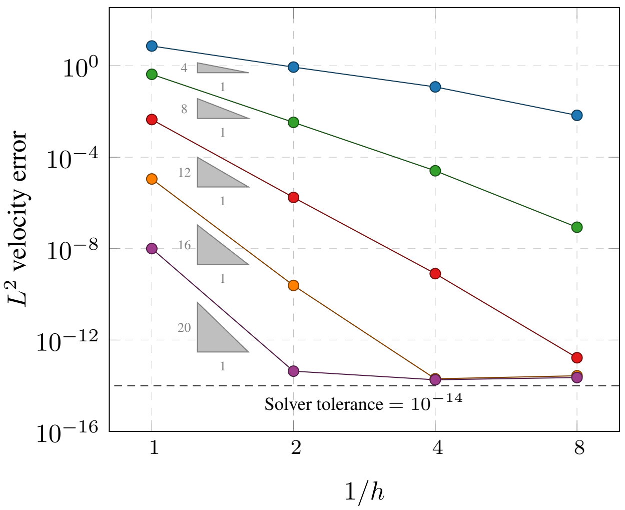


Figure 4.3: L^2 velocity error showing high-order spatial convergence for steady-state Stokes. Polynomial degrees $3, 7, 11, 15$, and 19 are used for the velocity finite element space.

condition number with increasing polynomial degree [58].

For the case of h-refinement, we fix the polynomial degree to be $p = 7$, and perform a sequence of uniform refinements. The initial mesh is a 4×4 Cartesian grid with 841 DoFs. We perform five refinements, so that the finest mesh is 128×128 with 804,609 DoFs. The number of iterations required to converge to tolerance is shown in Figure 4.2. Here, we observe approximately constant iterations, independent of the mesh refinement. These examples verify the robustness of the sub-problem preconditioners with respect to the mesh size and polynomial degree.

4.3.2 Steady-state Stokes flow

To verify the high-order accuracy of the spatial discretization, and to test the convergence properties of the solver, we solve the steady Stokes equations with a smooth solution in two spatial dimensions. Setting $\nu = 1$, we choose the right-hand side to be

$$\begin{aligned} f_1(x,y) &= \pi\cos(\pi y)\left(4\pi^2(1 - 2\cos(2\pi x))\sin(\pi y) - \sin(\pi x)\right), \\ f_2(x,y) &= 2\pi^3\sin(2\pi x)(2\cos(2\pi y) - 1) - \pi\cos(\pi x)\sin(\pi y). \end{aligned}$$

The exact solution is then given by

$$\begin{aligned} u_1(x,y) &= 2\pi\sin^2(\pi x)\sin(\pi y)\cos(\pi y), \\ u_2(x,y) &= -2\pi\sin(\pi x)\cos(\pi x)\sin^2(\pi y), \\ p(x,y) &= \cos(\pi x)\cos(\pi y). \end{aligned}$$

Table 4.1: Error and convergence results for steady Stokes equation, showing L^2 error norms for velocity and pressure, and number of FGMRES iterations required to reduce the residual by a factor of 10^{14}.

$p = 7$

$1/h$	$\|\boldsymbol{u}_h - \boldsymbol{u}\|_2$	Rate	$\|p_h - p\|_2$	Rate	Its.
1	4.25×10^{-1}	—	1.61×10^{-1}	—	31
4	3.36×10^{-3}	6.99	3.33×10^{-3}	5.60	41
16	2.55×10^{-5}	7.04	7.58×10^{-5}	5.46	43
64	8.63×10^{-8}	8.21	2.41×10^{-7}	8.30	46

$p = 11$

$1/h$	$\|\boldsymbol{u}_h - \boldsymbol{u}\|_2$	Rate	$\|p_h - p\|_2$	Rate	Its.
1	4.46×10^{-3}	—	1.72×10^{-2}	—	35
4	1.74×10^{-6}	11.32	1.33×10^{-6}	13.66	42
16	8.05×10^{-10}	11.08	2.57×10^{-9}	9.01	47
64	1.68×10^{-13}	12.23	6.76×10^{-13}	11.89	49

$p = 15$

$1/h$	$\|\boldsymbol{u}_h - \boldsymbol{u}\|_2$	Rate	$\|p_h - p\|_2$	Rate	Its.
1	1.12×10^{-5}	—	1.05×10^{-2}	—	39
4	2.44×10^{-10}	15.48	1.46×10^{-10}	26.10	45
16	2.01×10^{-14}	13.57	5.30×10^{-13}	8.11	52
64	2.73×10^{-14}	-0.44	7.43×10^{-13}	-0.49	53

$p = 19$

$1/h$	$\|\boldsymbol{u}_h - \boldsymbol{u}\|_2$	Rate	$\|p_h - p\|_2$	Rate	Its.
1	1.00×10^{-8}	—	7.24×10^{-3}	—	41
4	4.34×10^{-14}	17.82	3.22×10^{-13}	34.39	47
16	1.82×10^{-14}	1.25	7.48×10^{-13}	-1.21	52
64	2.29×10^{-14}	-0.33	8.69×10^{-13}	-0.22	55

The exact solution is imposed as a Dirichlet boundary condition for velocity on all domain boundaries using the method of manufactured solutions. We run this case with polynomial degrees from $p = 3$ to $p = 19$ on a sequence of uniformly refined Cartesian meshes. We solve the resulting linear system using FGMRES with the matrix-free block triangular preconditioners described in Section 4.2.4. The stopping criterion for the iterative solver is a relative residual norm of 10^{-14}. The spatial convergence is shown in Figure 4.3. The expected $p+1$ order of accuracy was observed in all cases, verifying the high-order spatial accuracy of the underlying matrix-free discretization. In Table 4.1, we show the L^2 error for velocity and

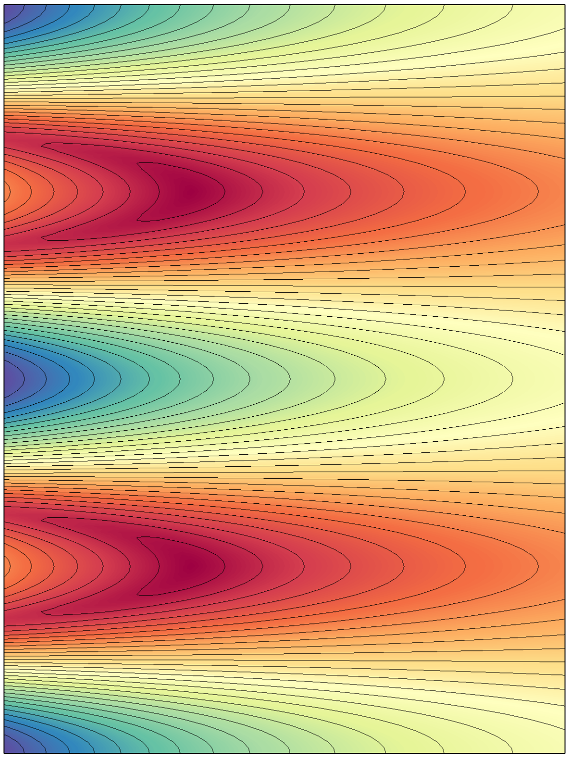


Figure 4.4: Two-dimensional Kovasznay flow, showing contours of velocity magnitude computed using a coarse mesh with degree 11 polynomials.

pressure for cases considered, together with the number of FGMRES iterations required to converge the solution. The number of iterations shows a slight pre-asymptotic increase, but remains bounded with respect to both h and p.

4.3.3 Incompressible Navier-Stokes: Kovasznay flow

In 1948, Kovasznay presented an analytical solution to the stationary Navier-Stokes equations (2.11) in two spatial dimensions [76]. This solution may be used to represent the wake behind a periodic array of cylinders in the y-direction. The solution is given by

$$\begin{aligned} \lambda &= \frac{\mathrm{Re}}{2} - \sqrt{\frac{\mathrm{Re}^2}{4} + 4\pi^2}, \\ u_1(x,y) &= 1 - \exp(\lambda x)\cos(2\pi y), \\ u_2(x,y) &= \frac{\lambda}{2\pi}\sin(2\pi x), \\ p(x,y) &= -\frac{1}{2}\exp(2\lambda x), \end{aligned}$$

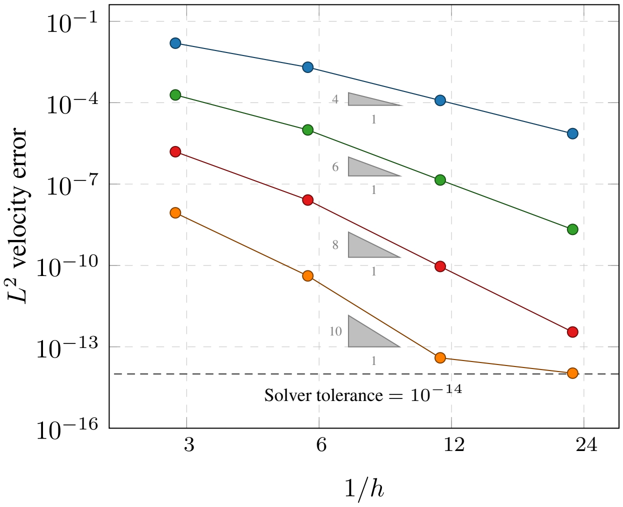


Figure 4.5: Two-dimensional Kovasznay flow. L^2 velocity error using polynomial degrees $p = 3, 5, 7, 9$.

where Re is the Reynolds number for the flow. In our example we define the problem in the rectangular domain $[-1/2, 1] \times [-1/2, 3/2]$ with Re = 40. Velocity magnitude contours of the solution are shown in Figure 4.4. We use pseudo-time integration in order to apply the projection method described in Section 4.1.2 with BDF2 to the steady-state problem. The pseudo-time step is chosen to be $\Delta t = 10^{-3}$ on the coarsest mesh, and is reduced by a factor of two with each refinement. The exact solution is enforced as a Dirichlet boundary condition on all boundaries of the domain. The equations are integrated until a final time of $t = 8$ to allow for the errors to propagate out of the domain.

To investigate spatial convergence we compute the solution with increasing polynomial degree and uniform spatial refinement. The L^2 errors for the velocity are shown in Figure 4.5. We observe the desired order of accuracy for all of the cases considered.

4.4 Summary

In this chapter, we have developed a suite of matrix-free linear solvers of high-order methods for incompressible flow. The foundational sub-problem solvers of the mass, Poisson, and Helmholtz systems do not require the costly assembly of high-order system matrices. Their memory usage is optimal, requiring only constant memory per degree of freedom. Furthermore, the number of operations required to apply the preconditioners scales linearly with the number of degrees of freedom, which is the same as the operator evaluation. The robustness of preconditioners with respect to the mesh size, polynomial degree, and time step was demonstrated on a range of test problems. Moreover, we considered a variety of incompressible problems, verifying the high-order accuracy and efficiency of the underlying method.

Chapter 5

Iterative subregion correction preconditioners with adaptive tolerance for problems with geometrically localized stiffness

In this chapter, we consider a particular class of problems exhibiting geometrically localized stiffness, defined by convergence rates of iterative methods degrading in a localized subregion of the mesh. On meshes with highly anisotropic elements, for instance those required by wall-resolved LES (WRLES), generic preconditioners can perform poorly on our Galerkin-discretized linear systems (2.53) [105]. We shall fully explore this breakdown for a block Jacobi stationary iterative method in the beginning of Section 5.6, but suffice to say the anisotropic elements converge slower than isotropic elements. A similar degradation in solver performance can be seen when solving a convection-diffusion problem with a local spike in diffusion. On the elements with large diffusion, the problem is stiffer than on the rest of the mesh, leading to slow iterative solver convergence rates. Likewise, radiative transfer through media with different material properties may introduce regions through which numerical methods suffer [88].

One option to avoid geometrically localized stiffness for a WRLES is to refine the mesh with a strategy that enforces a maximum allowable element aspect ratio [91]. While this strategy improves the performance of a preconditioner, it can also dramatically increase the total number of elements required to resolve the boundary layer, leading to costly solves. Another option to tackle geometrically localized stiffness is to develop more advanced preconditioners that robustly handle both diffusion-dominated and convection-dominated problems. A good solver should be robust with respect to anisotropic discretization, mesh parameters, and variable diffusivity. One approach was an aggregation-based algebraic multigrid method (AMG) [93] introduced into the AGMG software. A more recent approach was ℓAIR [85], which locally approximates the ideal restriction operator of AMG. However, both of these methods by design treat the entire domain simultaneously, leading to unnecessarily expensive iterations in the case of a small subregion of elements causing localized stiffness. Moreover, they both require access to the underlying matrix entries, so neither is applicable to matrix-free methods.

In this chapter, we derive a novel class of preconditioners that combines a domain decomposition methodology with the flexibility of choosing effective preconditioners for the overall problem and subregion problem. Specifically, we correct a globally preconditioned vector by the subregion-local error to ensure the resulting preconditioned vector is independent of the subregion stiffness. The subregion error equation is approximately solved by a preconditioned iterative method only to the same precision as the globally preconditioned vector to adaptively minimize subregion-local computations. Thus, subregion-local work is minimized while guaranteeing convergence rates. This strategy means that domain-specific knowledge can be leveraged to create an iterative subregion correction preconditioner that is robust and performant both across the overall domain and in the subregion. Moreover, we can combine matrix-free sub-preconditioners to create iterative subregion correction preconditioners for matrix-free methods.

5.1 Exact subregion correction preconditioner

We first motivate our preconditioners by presenting an exact subregion correction preconditioner. Consider the mesh partition,

$$\mathcal{T} = \mathcal{T}_{sr} \cup \mathcal{T}_{c}, \tag{5.1}$$

where $\mathcal{T}_{sr}$ is the subregion containing geometrically localized stiffness and $\mathcal{T}_{c}$ is its complement. On this partition, our discretization matrix is expanded as

$$A = \begin{pmatrix} A_{sr,sr} & A_{sr,c} \\ A_{c,sr} & A_{c,c} \end{pmatrix}. \tag{5.2}$$

Throughout this chapter, we subscript all vectors and matrices corresponding to the elements in the subregion and its complement with sr and c, respectively. So x_{sr} is the restriction of x to the subregion, an operation alternatively described as applying a rectangular restriction matrix to the global vector in domain decomposition literature [128].

Take P^{-1} to be a global preconditioner that does not couple degrees of freedom across elements. This property is satisfied for instance by one iteration of the block Jacobi method. While P^{-1} may perform well in the complement, the geometrically localized stiffness in the subregion prevents an overall good convergence rate. We define the exact subregion correction preconditioner as the block Gauss-Seidel type preconditioner,

$$\begin{pmatrix} A_{sr,sr} & A_{sr,c} \\ 0 & P_{c,c} \end{pmatrix} \begin{pmatrix} x_{sr} \\ x_{c} \end{pmatrix} = \begin{pmatrix} b_{sr} \\ b_{c} \end{pmatrix}, \tag{5.3}$$

where b is the input vector, and $x = (x_{sr}, x_c)^T$ is the final preconditioned vector. Defining the preliminary solution $\tilde{x} = P^{-1}b$, and noting that P^{-1} is element-wise block-diagonal, we can see that

$$\tilde{x}_c = \left(P^{-1}b\right)_c = P_{c,c}^{-1}b_c = x_c. \tag{5.4}$$

Therefore, by construction, (5.3) does not correct the preliminary solution in the complement of the subregion.

Moreover, the corrected solution given by (5.3), restricted to the subregion, is exactly

$$\begin{aligned} x_{sr} &= A_{sr,sr}^{-1}\left(b_{sr} - A_{sr,c}\tilde{x}_c\right) \\ &= \tilde{x}_{sr} + A_{sr,sr}^{-1}\left(b_{sr} - A_{sr,sr}\tilde{x}_{sr} - A_{sr,c}\tilde{x}_c\right) \\ &= \tilde{x}_{sr} + A_{sr,sr}^{-1}\left(b - A\tilde{x}\right)_{sr}. \end{aligned} \tag{5.5}$$

This rewriting provides an algorithm for directly computing the exact subregion correction preconditioner, presented as Algorithm 3.

Algorithm 3 Exact subregion correction preconditioner

```
Inputs: $b$ is the input vector, $k_{sr}$ is the list of elements in subregion
  function ExactPrecond($b$, $k_{sr}$)
      $\tilde{x} \leftarrow P^{-1}b$
      $r \leftarrow b - A\tilde{x}$
      $r_{sr} \leftarrow$ ExtractSubvector($r$, $k_{sr}$)
      $e_{sr} \leftarrow A_{sr,sr}^{-1}r_{sr}$
      $e \leftarrow$ PadSubvector($e_{sr}$, $k_{sr}$)
      $x \leftarrow \tilde{x} + e$
      return $x$
  end function
```

The first step is to apply a global preconditioner to the original right-hand side vector to create a preliminary solution $\tilde{x}$. After computing the global residual, r, the next step is to extract the residual corresponding to the elements in the subregion, r_{sr}. Next, the subregion error equation $A_{sr,sr}e_{sr} = r_{sr}$ is exactly solved for e_{sr}. This subregion error e_{sr} is padded with zeros outside the subregion to yield a global update e. Finally, the preliminary solution $\tilde{x}$ is corrected by e to yield the final preconditioned vector x.

This basic preconditioner has an attractive convergence property. If we assume that the subregion is decoupled from the rest of the mesh, then there is no interaction between the elements inside and outside the subregion. Therefore, matrix-vector multiplication and restriction to a sub-vector are commutative:

$$(Ax)_{sr} = A_{sr,sr}x_{sr}. \tag{5.6}$$

If this assumption is true, then our preconditioned vector satisfies

$$\begin{aligned} x_{sr} &= \tilde{x}_{sr} + A_{sr,sr}^{-1} \left(b - A\tilde{x}\right)_{sr} \\ &= \tilde{x}_{sr} + A_{sr,sr}^{-1} b_{sr} - \tilde{x}_{sr} \\ &= A_{sr,sr}^{-1} b_{sr}, \end{aligned} \tag{5.7}$$

within the subregion. That is, a solver built around this preconditioner will have convergence properties independent of the subregion. However, since assumption (5.6) does not hold in general, we revisit this convergence property in detail in Section 5.4. We also note that this exact preconditioner (5.3) can be considered a multiplicative Schwarz preconditioner, so the convergence properties of subspace-correction methods [60] apply.

It is also interesting to compare Algorithm 3 to the two-level multigrid method [22] used as a preconditioner [75]. Designed to solve linear systems from problems with frequency-localized stiffness, the two-level method works on a coarse grid in order to more efficiently solve the original, fine-grid problem. Not only does the exact subregion correction preconditioner visually appear similar to the two-level method without post-relaxation, but they are motivated similarly as well. By spending more effort on the cheaper problem, we seek to avoid spending more iterations on the expensive, full domain.

5.2 Iterative subregion correction preconditioner

We now describe the general iterative subregion correction preconditioner in Algorithm 4.

Algorithm 4 Iterative subregion correction preconditioner

```
Inputs: $b$ is the input vector, $k_{sr}$ is the list of elements in subregion
  function ApplyPrecond($b$, $k_{sr}$)
      $\tilde{x} \leftarrow P^{-1}b$
      $r \leftarrow b - A\tilde{x}$
      $r_{sr} \leftarrow$ ExtractSubvector($r$, $k_{sr}$)
      tol $\leftarrow$ AdaptiveTolerance($r$, $k_{sr}$)
      $e_{sr} \leftarrow$ GMRES($A_{sr,sr}$, $r_{sr}$, tol)        ▷ Solve subregion error equation $A_{sr,sr}e_{sr} = r_{sr}$
      $e \leftarrow$ PadSubvector($e_{sr}$, $k_{sr}$)
      $x \leftarrow \tilde{x} + e$
      return $x$
  end function
```

The only difference between this preconditioner and the exact version previously presented in Section 5.1 is its treatment of the subregion error equation. The subregion error equation is now only approximately solved by an inner iterative method to a requested tolerance. With a loose required inner tolerance, the approximate subregion solve could be much cheaper than any exact solve. Section 5.3 will show how we use r to adaptively compute a loose tolerance for the subregion-local solve.

This preconditioner has several attractive properties. First of all, it inherits the crucial subregion-independent convergence property from the exact subregion correction preconditioner as tol → 0. We will further analyze this convergence for inexact solves in Section 5.4.

The first step of this algorithm is to apply a global preconditioner to the original right-hand side vector to create a preliminary solution $\tilde{x}$. In principle, this could be any preconditioner, allowing one to leverage domain knowledge and use an effective preconditioner for the underlying problem that does not exhibit geometrically localized stiffness. Likewise, the subregion-local solve should be preconditioned as well, and a different preconditioner may prove to be better suited to the subregion problem. In practice, we have tested a block Jacobi method [116], a block incomplete LU (ILU) factorization method with no-fill [117, 105], and an algebraic multigrid (AMG) preconditioner [50, 137], all showing similar properties.

This preconditioner also has the attractive property that it requires only one global matrix-vector product beyond any generic preconditioner. All other work is local to the sub-operator and sub-vectors corresponding to the subregion. Therefore, for problems with a small number of elements in the subregion relative to the total number of elements in the overall mesh, this preconditioner is cheap. Indeed, this element fraction is a crucial factor in understanding the performance of our preconditioner, and we will explore its effects on performance for test problems in Section 5.6 and practical problems in Chapter 6.

Lastly, the only information this preconditioner requires beyond any generic preconditioner is the ability to extract a sub-vector from the global vector and vice-versa. This can be implemented simply by slicing vectors according to the list of elements composing the subregion: k_{sr}. Indeed, we shall show in Section 5.5 how this overall preconditioner is matrix-free if the global preconditioner and subregion-local preconditioner are matrix-free.

5.3 Adaptive tolerance selection

As described so far, the cost of the iterative subregion correction preconditioner is highly dependant on the cost of solving the subregion error equation $A_{sr,sr}e_{sr} = r_{sr}$. However, because the error correction in Algorithm 4 is local to the subregion, this local error equation does not call for an exact solution. In fact, we should optimally solve this local error equation only to the same precision that the global preconditioner "solved" the original problem outside the subregion in $\tilde{x}$. This idea forms the basis of the adaptive tolerance,

$$\mathrm{tol} = \Lambda \frac{\max_{k \notin k_{sr}} \|r_k\|_2}{\max_{k \in k_{sr}} \|r_k\|_2}, \tag{5.8}$$

where r_k refers to the element-wise residual (or sub-vector containing only the degrees of freedom corresponding to element k), and Λ is a scaling factor that approximately transforms operations on r to operations on $M^{-1}r$. In particular, we consider

$$\Lambda \leftarrow \frac{\left\| M_{c,c}^{-1} \right\|_2}{\left\| M_{sr,sr}^{-1} \right\|_2}. \tag{5.9}$$

In practice, this constant should be computed in a pre-processing step only once. Moreover, each 2-norm can be efficiently computed with 10 inverse iterations on $M_{sr,sr}$ and $M_{c,c}$, the restriction of the mass matrix to the subregion and its complement, respectively. Thus, the pre-processing required to compute this adaptive tolerance is cheap and element-wise local.

This adaptive tolerance is chosen to be the ratio of the maximum error found in the complement of the subregion and the maximum error found in the subregion. Therefore, the relative tolerance is the amount that the worst element inside the subregion must improve in order to catch up to the accuracy of the worst element outside the subregion. Since (5.8) only depends upon the residual r, which is already computed, this adaptive tolerance seamlessly integrates into the larger iterative subregion correction preconditioner. Consequently, solving the subregion error equation with an inner iterative method to this relative tolerance minimizes the cost of the overall preconditioner.

Residual vectors contain mesh mapping dependant scaling. The point of multiplying by this scaling factor Λ is to cheaply eliminate this mesh scaling, without having to compute a mass-inverse each outer iteration. Since $M^{-1}r$ does not contain these mesh scaling factors, we can use it as a scaled form of error. On meshes containing highly anisotropic elements within the subregion, the element-wise errors can be orders of magnitude larger than the element-wise residuals, necessitating errors be used.

This scaling factor Λ is intentionally chosen to be pessimistic, so that we can guarantee subregion-independent convergence of the overall preconditioner, as shown in Proposition 5.4.5. We have experimentally found that a more performant scaling factor is

$$\Lambda \leftarrow \min\left\{1, \frac{\left\| M_{i_c,i_c}^{-1} \right\|_2}{\left\| M_{i_{sr},i_{sr}}^{-1} \right\|_2}\right\}, \tag{5.10}$$

where i_{sr} and i_c are the elements inside the subregion and its complement at which the maxima computed in (5.8) are achieved. This choice results in a looser tolerance for the inner iterative solver, thus reducing the number of inner iterations required to solve the subregion error equation. However, the convergence analysis we develop in Section 5.4 does not directly apply to this choice of scaling factor, and so we do not present a guarantee that the overall solver will converge at a subregion-independent rate. However, this can be a worthwhile trade-off; we shall show in Section 5.6.1 that this adaptive tolerance reduces the required number of inner GMRES iterations considerably, resulting in overall computational savings. This performant scaling factor (5.10) depends on r and so now must be computed each outer iteration. In our implementation, we precompute all $\left\| M_{k,k}^{-1} \right\|_2$, reducing the online computation each iteration to a scalar division. We present our performant adaptive tolerance selection as Algorithm 5.

Algorithm 5 Adaptive tolerance selection

```
Inputs: $r$ is the global residual, $k_{sr}$ is the list of elements in subregion
  function AdaptiveTolerance($r$, $k_{sr}$)
    $(i_{sr}, e_{\text{max sr}}) \leftarrow \max_{k \in k_{sr}} \|r_k\|_2$
    $(i_c, e_{\text{max c}}) \leftarrow \max_{k \notin k_{sr}} \|r_k\|_2$
    $\Lambda \leftarrow \left\|M^{-1}_{i_c,i_c}\right\|_2 / \left\|M^{-1}_{i_{sr},i_{sr}}\right\|_2$        ▷ Scaling factor from residual to error
    tol $\leftarrow \Lambda \left(e_{\text{max c}}/e_{\text{max sr}}\right)$        ▷ Ratio of errors outside vs inside the subregion
    return tol
  end function
```

If desired, both of these scaling factors can be computed in a matrix-free manner. Since M is an element-wise block-diagonal matrix, the 2-norm decomposes into element-wise 2-norms. Each of these element-wise 2-norms can be computed with inverse iterations, the crucial step of which is the computation $M^{-1}_{k,k}b$. Using the matrix-free mass preconditioner developed in Section 4.2.1, these element-wise inverses can be computed efficiently. Thus, the pre-processing of all $\left\|M^{-1}_{k,k}\right\|_2$ is matrix-free, and so (5.9) or (5.10) is too.

Finally, we note that the scaling factor is only important if the mesh elements are highly graded. If the elements across the mesh are mostly the same size, this step is irrelevant, and we could use element-wise residuals directly instead by choosing $\Lambda \leftarrow 1$.

5.4 Convergence analysis

Let us return to the basic convergence analysis presented in (5.7) for the exact subregion correction preconditioner. We now consider the general case, where assumption (5.6) does not hold. We are interested in quantifying the error of our preconditioner, defined as

$$\Delta = x^* - x, \tag{5.11}$$

where x^* is the true solution, satisfying $Ax^* = b$. The preliminary error, defined as

$$\tilde{\Delta} = x^* - \tilde{x}, \tag{5.12}$$

is useful in this analysis, where $\tilde{x}$ is the preliminary solution.

Proposition 5.4.1. *The error of our preconditioner inside the subregion relates to the preliminary error outside the subregion according to*

$$\Delta_{sr} = -A^{-1}_{sr,sr}A_{sr,c}\tilde{\Delta}_c. \tag{5.13}$$

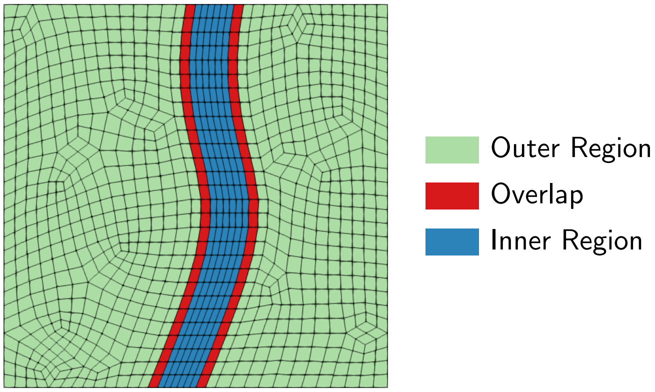


Figure 5.1: Schematic image of a mesh partitioned into an outer region, inner region, and overlapping region of elements. The geometrically localized stiffness is contained within the inner region, so the subregion is made up of elements in the inner region and overlap.

Proof. The true solution satisfies $Ax^* = b$. Therefore, expanding according to (5.2), the true solution restricted to the subregion satisfies

$$x^*_{sr} = A^{-1}_{sr,sr} \left(b_{sr} - A_{sr,c} x^*_c \right). \tag{5.14}$$

Subtracting (5.5) from (5.14), we have

$$\Delta_{sr} = A^{-1}_{sr,sr} A_{sr,c} \left(\tilde{x}_c - x^*_c \right) = -A^{-1}_{sr,sr} A_{sr,c} \tilde{\Delta}_c \tag{5.15}$$

□

Remark. *This result establishes that the error inside the subregion Δ_{sr} is directly related to the preliminary error outside the subregion $\tilde{\Delta}_c$. The remainder of this analysis is devoted to finding bounds on these scaling matrices and the preliminary error.*

Now consider a mesh $\mathcal{T}_3$ partitioned into three regions: an inner region that demonstrates geometrically localized stiffness, an outer region free from this stiffness, and a third region between the two regions that is one element thick. We write this partition as

$$\mathcal{T}_3 = \mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{over}} \cup \mathcal{T}_{\text{out}}, \tag{5.16}$$

and present an example in Figure 5.1. Region $\mathcal{T}_{\text{in}}$ has a stiffness parameter ε (larger ε is stiffer), which generally could represent diffusion coefficient, reciprocal mesh size, etc. However, regions $\mathcal{T}_{\text{over}}$ and $\mathcal{T}_{\text{out}}$ locally do not depend on ε, for instance by having a fixed diffusion coefficient much smaller than ε. We must determine how to choose $\mathcal{T}_{sr}$ from these regions such that Δ_{sr} is ε-independent, so that a solver preconditioned with iterative subregion correction should have a ε-independent convergence rate.

Proposition 5.4.2. *Let A be the matrix associated with our discretization on this mesh $\mathcal{T}_3$. Then the blocks of the partitioned A corresponding to $\mathcal{T}_{\text{out}}$ are independent of this stiffness parameter ε.*

Proof. Because our underlying spatial discretization (2.27) is compact, only the degrees of freedom belonging to neighboring elements are connected by the discretization. Therefore the degrees of freedom in the inner region are not connected to those of the outer region, resulting in zero blocks in the matrix expansion below.

$$A = \begin{pmatrix} A_{\text{in,in}} & A_{\text{in,over}} & 0 \\ A_{\text{over,in}} & A_{\text{over,over}} & A_{\text{over,out}} \\ 0 & A_{\text{out,over}} & A_{\text{out,out}} \end{pmatrix}. \tag{5.17}$$

From (5.17), the block rows $A_{\text{out},j}$ and columns $A_{i,\text{out}}$ only relate $\mathcal{T}_{\text{over}}$ and $\mathcal{T}_{\text{out}}$ and thus have no dependence on ε. □

Remark. *An important conclusion of this proposition is that $\mathcal{T}_{\text{out}}$ is completely independent of ε. The same cannot be said about the region $\mathcal{T}_{\text{over}}$ because the compact discretization couples $\mathcal{T}_{\text{over}}$ with $\mathcal{T}_{\text{in}}$, so both $A_{\text{in,over}}$ and $A_{\text{over,in}}$ depend on ε.*

Remark. *Technically, this result does not require a compact discretization. This argument only relies upon the discretization matrix not coupling ε-dependence across non-neighboring elements.*

Therefore, we should define the subregion as

$$\mathcal{T}_{sr} := \mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{over}}. \tag{5.18}$$

It should now be clear why we label this third region an "overlap"—it has the properties of the outer region but is part of our subregion. With this choice of subregion, Proposition 5.4.2 shows that a block Jacobi method applied to $\mathcal{T}_{\text{out}}$ will result in a preliminary error that is ε-independent. All that remains is to show the scaling matrices in (5.13) do not introduce any ε-dependence. However, to show this, we first make the following assumption:

$$\left\| M_{sr,sr}^{-1} A_{sr,sr} \right\| \geq C_1, \tag{5.19}$$

where $M_{sr,sr}^{-1}$ is the inverse mass matrix restricted to the subregion, and C_1 is independent of ε.

We note that in the case of increasing diffusivity, this is a reasonable assumption since $\|A_{sr,sr}\|$ remains uniformly bounded from below; only the upper bound increases. However, this lower bound is not uniform when ε relates to decreasing mesh size. Because the elemental volume integrals have a scaling of h^d, the lower bound for $\|A_{sr,sr}\|$ decreases with decreasing h (increasing ε). The scaled matrix $M_{sr,sr}^{-1} A_{sr,sr}$ on the other hand has a uniform bound, independent of mesh size [3]. Therefore, (5.19) is a reasonable additional assumption to require for the following proposition.

Moreover, we are focused on the solution of time-dependent problems yielding linear systems (2.53). For any positive-definite matrix J, the matrix

$$M^{-1}A = M^{-1}(M - \Delta t J) = I - \Delta t M^{-1} J \tag{5.20}$$

naturally has a minimum eigenvalue of 1. Therefore, (5.19) is especially natural for the time-dependent problems we are solving, even though our analysis considers the more general case of indefinite J.

Proposition 5.4.3. *Let Δ_{sr} and $\tilde{\Delta}_c$ be the error inside the subregion and the preliminary error outside the subregion, respectively. Choosing the subregion on mesh $\mathcal{T}_3$ according to* (5.18) *and assuming* (5.19)*, we have*

$$\|\Delta_{sr}\| \leq C \left\|\tilde{\Delta}_c\right\|, \tag{5.21}$$

for C independent of ε.

Proof. From our discretization on $\mathcal{T}_3$ we have

$$A_{sr,c} = \begin{pmatrix} A_{\text{in,out}} \\ A_{\text{over,out}} \end{pmatrix} = \begin{pmatrix} 0 \\ A_{\text{over,out}} \end{pmatrix}. \tag{5.22}$$

Because $A_{\text{over,out}}$ is independent of ε by construction, so is $A_{sr,c}$. We therefore also have

$$\left\|M_{sr,sr}^{-1} A_{sr,c}\right\| \leq C_2, \tag{5.23}$$

for C_2, a constant independent of ε. From Proposition 5.4.1, we have

$$\begin{aligned} \|\Delta_{sr}\| &= \left\|A_{sr,sr}^{-1} A_{sr,c} \tilde{\Delta}_c\right\| \\ &= \left\|A_{sr,sr}^{-1} M_{sr,sr} M_{sr,sr}^{-1} A_{sr,c} \tilde{\Delta}_c\right\| \\ &= \left\|\left(M_{sr,sr}^{-1} A_{sr,sr}\right)^{-1} \left(M_{sr,sr}^{-1} A_{sr,c}\right) \tilde{\Delta}_c\right\|. \end{aligned} \tag{5.24}$$

Our assumption (5.19) provides a lower bound on $\left\|M_{sr,sr}^{-1} A_{sr,sr}\right\|$, so the inverse is bounded above:

$$\left\|\left(M_{sr,sr}^{-1} A_{sr,sr}\right)^{-1}\right\| \leq C_1. \tag{5.25}$$

Combining (5.25) and (5.23), we can estimate $\|\Delta_{sr}\|$.

$$\|\Delta_{sr}\| \leq \left\|\left(M_{sr,sr}^{-1} A_{sr,sr}\right)^{-1}\right\| \left\|M_{sr,sr}^{-1} A_{sr,c}\right\| \left\|\tilde{\Delta}_c\right\| \leq C \left\|\tilde{\Delta}_c\right\|, \tag{5.26}$$

for $C = C_1 C_2$ independent of ε. □

Remark. *This key result, combined with our previous conclusion from Proposition 5.4.2 shows that under a few reasonable assumptions, the convergence rate of a solver preconditioned with iterative subregion correction is expected to be independent of ε.*

We will confirm that this ε-independence convergence holds in practice for the test problems we model in Section 5.6, where each subregion is composed of contiguous elements containing the geometrically localized stiffness and at least one further element in each direction, exactly like $\mathcal{T}_3$.

If we were to not include the overlap region as part of the subregion and instead choose $\mathcal{T}_{sr} := \mathcal{T}_{\text{in}}$, we would lose this key ε-independence. This is because $\mathcal{T}_{\text{over}}$ is coupled to $\mathcal{T}_{\text{in}}$, and so the error in $\mathcal{T}_{\text{over}}$ after one application of block Jacobi iteration would increase as ε increases. Since a subset of $\tilde{\Delta}_c$ is ε-dependent, we should expect ε-dependent convergence rates by Proposition 5.4.3.

Finally, we can incorporate the adaptive tolerance defined by (5.8) and (5.9) to analyze convergence of the iterative subregion correction preconditioners.

Proposition 5.4.4. *When only solving to the tolerance chosen by (5.8) and (5.9), the error of our preconditioner inside the subregion relates to the preliminary error outside the subregion according to*

$$\Delta_{sr} = A_{sr,sr}^{-1}\left(M_{sr,sr}w_{sr} + A_{sr,c}\tilde{\Delta}_c\right), \tag{5.27}$$

where $\|w_{sr}\|_2 \leq C_3 \left\|M_{c,c}^{-1}\right\|_2 \|r_c\|_2$ for a constant C_3.

Proof. For a solver converging to the adaptive tolerance chosen by (5.8), we have

$$\|r_{sr} - A_{sr,sr}e_{sr}\|_2 \leq \text{tol}\,\|r_{sr}\|_2\,. \tag{5.28}$$

We note that the mass matrix is block-diagonal, where one block corresponds to one element. Therefore, the inverse mass matrix is likewise block-diagonal, with no communication across elements. Breaking apart (5.28) into the local element-wise ℓ^2-norms, we have

$$\left\|(r_{sr} - A_{sr,sr}e_{sr})_k\right\|_2 \leq \text{tol}\,\|r_k\|_2 \quad \forall k \in k_{sr}. \tag{5.29}$$

We again subscript a vector by k to denote that the sub-vector contains only the degrees of freedom corresponding to element k. For Λ defined by (5.9), (5.29) expands to

$$\left\|(r_{sr} - A_{sr,sr}e_{sr})_k\right\|_2 \leq \max_{k\notin k_{sr}} \|r_k\|_2 \left(\frac{\left\|M_{c,c}^{-1}\right\|_2}{\left\|M_{sr,sr}^{-1}\right\|_2}\right) \quad \forall k \in k_{sr}. \tag{5.30}$$

Recombining these sub-vectors yields

$$\|r_{sr} - A_{sr,sr}e_{sr}\|_2 \leq C_3 \left(\frac{\left\|M_{c,c}^{-1}\right\|_2}{\left\|M_{sr,sr}^{-1}\right\|_2}\right) \|r_c\|_2\,, \tag{5.31}$$

where this constant C_3 is loosely bounded for instance by the number of elements in the mesh. Expressing (5.31) another way, there exists a vector $v_{sr} \in \mathbb{R}^{|k_{sr}|}$ such that

$$r_{sr} = A_{sr,sr} e_{sr} + v_{sr}, \tag{5.32}$$

where

$$\|v_{sr}\|_2 \leq C_3 \left(\frac{\left\|M_{c,c}^{-1}\right\|_2}{\left\|M_{sr,sr}^{-1}\right\|_2} \right) \|r_c\|_2 \,. \tag{5.33}$$

Choose $w_{sr} = M_{sr,sr}^{-1} v_{sr}$. Returning to the preconditioner analysis in (5.5), we now have

$$\begin{aligned} x_{sr} &= \tilde{x}_{sr} + A_{sr,sr}^{-1} \left((b - A\tilde{x})_{sr} - M_{sr,sr} w_{sr} \right) \\ &= \tilde{x}_{sr} + A_{sr,sr}^{-1} \left(b_{sr} - A_{sr,sr}\tilde{x}_{sr} - A_{sr,c}\tilde{x}_c - M_{sr,sr} w_{sr} \right) \\ &= A_{sr,sr}^{-1} \left(b_{sr} - A_{sr,c}\tilde{x}_c - M_{sr,sr} w_{sr} \right). \end{aligned} \tag{5.34}$$

Subtracting (5.34) from (5.14), we arrive at (5.27). □

Remark. *The only difference between this result and Proposition 5.4.1 is this extra w_{sr} error term. But since this term is small, we can now prove a similar result to Proposition 5.4.3.*

Proposition 5.4.5. *Let Δ_{sr} and $\tilde{\Delta}_c$ be the error inside the subregion and the preliminary error outside the subregion when only solving to the adaptive tolerance chosen by (5.8) and (5.9), respectively. Choosing the subregion on mesh $\mathcal{T}_3$ according to (5.18) and assuming (5.19), we have*

$$\|\Delta_{sr}\| \leq C_4 \|\delta_c\| + C \left\|\tilde{\Delta}_c\right\|, \tag{5.35}$$

for C_4, C constants independent of ε and $\|\delta_c\| = \left\|M_{c,c}^{-1}\right\| \|r_c\|$.

Proof. From Proposition 5.4.4, we can use the triangle inequality to get

$$\begin{aligned} \|\Delta_{sr}\| &= \left\| A_{sr,sr}^{-1} \left(M_{sr,sr} w_{sr} + A_{sr,c}\tilde{\Delta}_c \right) \right\| \\ &\leq \left\| A_{sr,sr}^{-1} M_{sr,sr} w_{sr} \right\| + \left\| A_{sr,sr}^{-1} A_{sr,c} \tilde{\Delta}_c \right\|. \end{aligned} \tag{5.36}$$

We reuse assumption (5.19) to provide the upper bound

$$\left\| \left(M_{sr,sr}^{-1} A_{sr,sr} \right)^{-1} \right\| \leq C_1. \tag{5.37}$$

Therefore the first term of (5.36) can be bounded by

$$\begin{aligned} \left\| A_{sr,sr}^{-1} M_{sr,sr} w_{sr} \right\| &\leq \left\| \left(M_{sr,sr}^{-1} A_{sr,sr} \right)^{-1} \right\| \left\| M_{sr,sr}^{-1} \right\| \|v_{sr}\| \\ &\leq C_1 C_3 \left\| M_{c,c}^{-1} \right\| \|r_c\| \\ &= C_4 \|\delta_c\| \,. \end{aligned} \tag{5.38}$$

Here we have introduced the constant $C_4 = C_1 C_3$ and a form of scaled error $\|\delta_c\| = \left\|M_{c,c}^{-1}\right\| \|r_c\|$. The second term of (5.36) is bounded by $C \left\|\tilde{\Delta}_c\right\|$ as shown in (5.26). Combining these two bounds results in (5.35). □

Remark. *This result is similar to Proposition 5.4.3 except with an extra error term. However, this extra error is bounded by an ε-independent quantity if we choose $\mathcal{T}_{sr} := \mathcal{T}_{\text{in}} \cup \mathcal{T}_{\text{over}}$ as before. This is because for this choice of subregion, $\|\delta_c\|$ is approximately the size of $\left\|\tilde{\Delta}_c\right\|$ and is in particular ε-independent. Therefore, for this choice of adaptive tolerance, we still recover the desired ε-independent convergence rate.*

We do not show a full convergence analysis for the more performant scaling factor Λ presented in (5.10) because this looser tolerance for the subregion error equation iterative solver can in fact result in slightly ε-dependent convergence rates. Specifically, this degradation occurs when the element in the subregion with the largest local residual is not the element in the subregion corresponding to the largest mesh mapping. For outer iterations when this is the case, the first term of (5.36) will not be bounded by $C_4 \|\delta_c\|$, but instead be bounded by

$$C_4 \frac{\left\|M_{sr,sr}^{-1}\right\|}{\left\|M_{i_{sr},i_{sr}}^{-1}\right\|} \|\delta_c\| \,, \tag{5.39}$$

a mesh-dependent bound. However, we have experimentally found that the drastic reduction in necessary inner iterations more than compensates for the slight increase in outer iterations, as show in Section 5.6.1. We reiterate that this concern is not relevant to mostly isotropic meshes where the geometrically localized stiffness is caused by a locally large diffusion coefficient ε.

5.5 Choice of subregion preconditioners

Algorithm 4 describes a preconditioner that depends upon a global (or outer) preconditioner and subregion-local (or inner) preconditioner. Indeed, this forms a class of preconditioners with performance dependant on the specific sub-preconditioners used to solve the original problem (2.53). As we shall see in Section 5.6, this flexibility allows one to leverage domain-specific knowledge of a given problem to create a specific iterative subregion correction preconditioner that is robust and performant globally and locally. We note that while the specific choice of sub-preconditioners should depend on the specific problem, the number of outer iterations will be independent of the number of inner iterations for any combination of preconditioners.

Unlike the exact subregion correction preconditioner, specific instances of iterative subregion correction preconditioners are nonlinear and therefore should be coupled with a flexible outer iterative method such as FGMRES [114]. Throughout Section 5.6, we will explore a few choices of outer and inner preconditioner such as block Jacobi [116], block ILU [117, 105], and AMG [50, 137].

Finally, we consider these preconditioners in the broader context of the matrix-free operators developed in Chapter 3. Every step described in Algorithm 4 is immediately matrix-free with the exception of applying the outer and inner preconditioners, and extracting $A_{sr,sr}$ from A. We could directly represent

$$A_{sr,sr} = R^T AR, \tag{5.40}$$

where R is the restriction matrix from the global domain to the subregion. While this construction is non-invasive, it has the disadvantage that computing the subregion-local matrix operations requires the full domain matrix. Therefore, we return to the CEED operator defined by (3.26). We define the invasive construction

$$A_{sr,sr} = \mathcal{P}^T \mathcal{E}_{sr}^T \mathcal{B}^T D \mathcal{B} \mathcal{E}_{sr} \mathcal{P}, \tag{5.41}$$

where the submatrix extraction is implemented by modifying the $\mathcal{E}$ operator to $\mathcal{E}_{sr}$. This construction exclusively extracts elements in the subregion, thus allowing for matrix-free subregion-local operations.

To create a fully matrix-free iterative subregion preconditioner, we require the outer and inner preconditioners to also be matrix-free preconditioners, such as those presented in Chapter 4. Furthermore, block Jacobi can be implemented as a matrix-free preconditioner [116]. One could also use more advanced matrix-free preconditioners [11, 56, 51], although we shall see that using more expensive preconditioners does not guarantee performance savings.

5.6 Numerical results

Now we explore convergence and performance results of several instances of our iterative subregion correction preconditioners applied to different problems. Throughout this section, we couple our adaptive tolerance with a fixed maximum number of inner iterations of 100. Moreover, we enforce that the absolute tolerance for the inner iterative method is no smaller than the tolerance from the outer iteration to avoid unnecessary inner iterations. To this end, we also do not run any inner iterations if the adaptive tolerance is greater than 1. While it is possible to choose different inner solver parameters that will outperform this selection of parameters depending on the specifics of a mesh or problem, we have found that our choices result in a performant solver that is robust across a wide array of problems.

The spatial discretizations used in this section were implemented with in-house software as well as MFEM [2]. The sub-preconditioners were implemented in-house as well, except for BoomerAMG which is implemented in *hypre* [50].

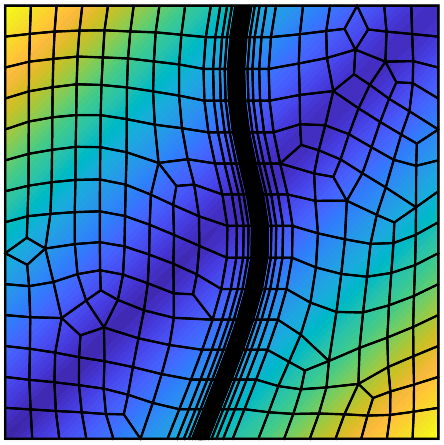


Figure 5.2: Steady state solution of convection-diffusion problem on a mesh containing anisotropic elements. The largest element aspect ratio is approximately 150 : 1.

5.6.1 Convection-diffusion, anisotropic mesh

In this section, we explore our solver's properties when applied to a problem with mesh-dependent geometrically localized stiffness. Specifically, we solve the 2D convection-diffusion equation (2.14), where $\mathbf{v}$ is a fixed velocity field $(1, 1)^T$ and $\varepsilon = 10^{-3}$ is a constant diffusion coefficient across the entire domain. This problem is solved on a mesh with highly anisotropic elements in the center of the mesh. The largest element aspect ratio is approximately 150 : 1. The steady state solution is overlaid on this mesh in Figure 5.2.

Choosing $\Delta t = 10^{-3}$, we can run a few experiments with this problem. First we run a stationary iteration preconditioned with block Jacobi on the trivial system $Ax = 0$. As described in Section 5.4, we have a mesh $\mathcal{T}_3$ partitioned into three regions: an inner region that demonstrates geometrically localized stiffness, an outer region free from this stiffness, and the overlap region between the two regions that is one element thick. The ℓ_∞ errors restricted to each of the three regions are plotted against iterations in the left half of Figure 5.3.

We note that the convergence rate in $\mathcal{T}_{\text{over}}$ is not independent of $\varepsilon \sim 1/h$, even in the first iteration. Because $\mathcal{T}_{\text{over}}$ and $\mathcal{T}_{\text{in}}$ are coupled, this is expected and matches our conclusions from Proposition 5.4.2. Meanwhile, the convergence rate in $\mathcal{T}_{\text{out}}$ is initially independent of ε, but the asymptotic rate of convergence lowers to that of $\mathcal{T}_{\text{in}}$. Because this problem is globally coupled, the asymptotic convergence rate is determined by the slow convergence in $\mathcal{T}_{\text{in}}$. As the mesh becomes increasingly anisotropic, the convergence rate will degrade in the entire mesh, including in the isotropic region $\mathcal{T}_{\text{out}}$.

The fact that regions $\mathcal{T}_{\text{in}}$ and $\mathcal{T}_{\text{out}}$ have the same asymptotic convergence rate but significantly different initial convergence rates is precisely the motivation for the iterative subregion correction preconditioners. We now repeat this experiment but apply a stationary iteration

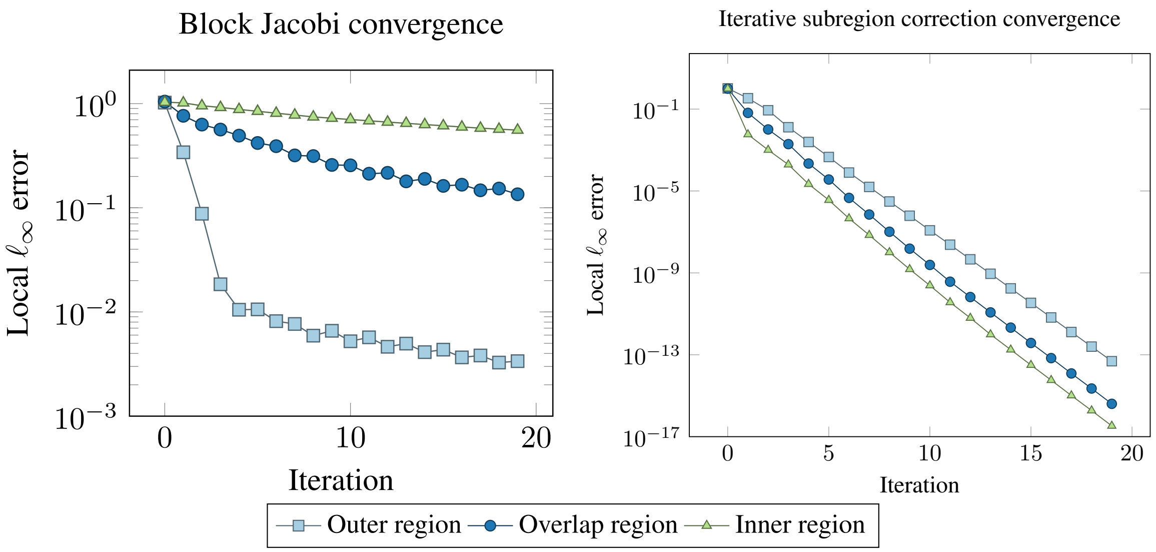


Figure 5.3: Comparison of local errors of block Jacobi and iterative subregion correction applied to convection-diffusion problem on an anisotropic mesh. The asymptotic rates of convergence for block Jacobi are the same as that of the inner region. In contrast, the asymptotic rates of convergence for iterative subregion correction are the same as that of the outer region in agreement with our conclusions from Proposition 5.4.3.

using the exact subregion correction method. The ℓ_∞ norms are shown in the right panel of Figure 5.3. Because the subregion operator is solved exactly, and in accordance with Proposition 5.4.3, the convergence rates in all regions are determined by the initial convergence rate in the isotropic region $\mathcal{T}_{\text{out}}$, which is independent of the mesh anisotropy. By performing extra work in the subregion, the errors in $\mathcal{T}_{\text{in}}$ and $\mathcal{T}_{\text{over}}$ now converge at the much faster rate of $\mathcal{T}_{\text{out}}$, and the solver achieves overall convergence independent of ε.

Next we analyze the cost of this preconditioner. However, because the overall cost of this class of preconditioners varies widely depending on implementation details of the various components of the algorithm, we introduce a performance model to compare the performance of different runs. The cost of the algorithm is dominated by the matrix-vector products (matvecs) performed in the outer and inner iterative methods. Therefore, a reasonable performance model counts the number of equivalent global matrix-vector products performed by the algorithm. Using FMGRES preconditioned by Algorithm 4, we see the method requires two global matrix-vector products per outer iteration, one application of the global preconditioner, and a subregion solve. Therefore, we define our performance model as

$$\text{cost} = (2 + \gamma_{\text{p outer}})\, n_{\text{outer}} + (1 + \gamma_{\text{p inner}})\, \frac{n_{sr}}{n_k} n_{\text{inner}}, \tag{5.42}$$

where $n_{sr} = |k_{sr}|$ is the number of elements in the subregion and n_k is the number of ele-

Solver configuration	n_{outer}	n_{inner}
Performant	6	195
Provable	6	246
Block Jacobi	99	—

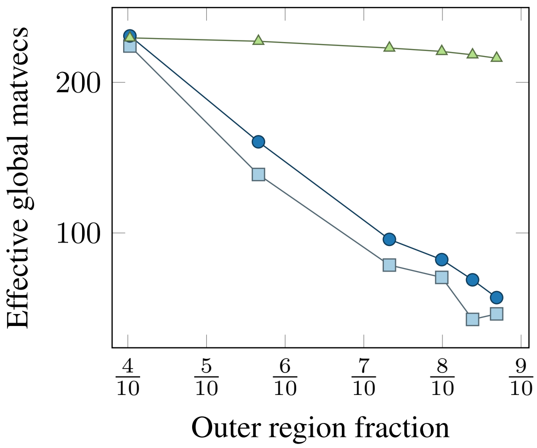


Figure 5.4: Effective global matrix-vector products required by FGMRES with three different preconditioners to converge to a tolerance of 10^{-8} for the convection-diffusion problem. Performance savings are more substantial as the outer region grows. For the 3/4 outer region problem, both iterative subregion correction preconditioners require only 6 outer iterations to converge, while block Jacobi requires 99 iterations.

ments across the domain. $\gamma_{\text{p outer}}$ and $\gamma_{\text{p inner}}$ are the weighting factors scaling from the cost of a matrix-vector product to the approximate cost of the outer and inner preconditioner, respectively. One application of the block Jacobi and block ILU preconditioners costs approximately 1/4 and 1 times the cost of a matrix-vector product, respectively. Empirically, we have found that AMG costs approximately 2 times the cost of a matrix-vector product. n_{outer} and n_{inner} are the number of inner and outer iterations, respectively.

To explore the performance of our preconditioner, we solve this same convection-diffusion problem to a relative outer tolerance of 10^{-8}, but on several domains of increasing size. Specifically, each successively larger problem adds elements to extend the domain (and outer region) but does not modify the inner or overlap regions. This experiment allows us to measure the performance of our method against the fraction of elements in the mesh that belong to the outer region,

$$\text{outer region fraction} = \left(1 - \frac{n_{sr}}{n_k}\right). \tag{5.43}$$

Figure 5.4 shows the performance model results using adaptive tolerance iterative subregion correction with the more performant scaling factor (5.10), adaptive tolerance iterative subregion correction with the provable scaling factor (5.9), and block Jacobi.

First of all, we see that iterative subregion correction results in computational savings compared to block Jacobi over all domains with more than 1/2 of the domain outside the subregion. Moreover, as expected, performance savings are more substantial as the outer region grows, with a 4.15× improvement reported for the 87/100 outer region fraction problem. This downward trend should only continue as the subregion fraction decreases further until the cost of the inner iterations at the subregion level is immaterial relative to the cost of the outer iterations. In this experiment, that horizontal asymptote occurs at 13.5 effective global matrix-vector products, which would be an approximate 16× improvement over pure block Jacobi. For all the runs with our iterative subregion correction preconditioner, FGMRES converges in exactly 6 iterations, again showing that our solver is ε-independent. Because the more performant scaling factor Λ from (5.10) needs less inner iterations to achieve the same overall accuracy, for this problem it is a strict improvement over the original scaling factor (5.9). Meanwhile, FGMRES converges in anywhere between 96 and 102 iterations when preconditioned with block Jacobi.

5.6.2 Convection-diffusion, variable diffusion coefficient

In this section, we explore the properties of a solver based on the iterative subregion correction preconditioner for a test problem for which the geometrically localized stiffness is not mesh-dependent. Again, we solve the 2D convection-diffusion equation (2.14), but now with a variable $\varepsilon(\boldsymbol{x})$ controlling the stiffness of the derived linear system. The background $\varepsilon = \varepsilon_{\text{out}}$ is fixed, and a subregion of the domain has a much larger diffusion coefficient at $\varepsilon = \varepsilon_{\text{in}} \gg \varepsilon_{\text{out}}$. This problem setup produces the steady-state solution shown in Figure 5.5.

For this problem, at $\Delta t = 10^{-3}$, an ILU preconditioner would be efficient across most of the domain (an advection-dominated problem), but GMRES preconditioned with a naive ILU preconditioner performs poorly because the elements with ε_{in} create a locally diffusion-dominated problem. We explore the performance of block ILU with no fill (BILU) to solve this problem, as well as two iterative subregion correction preconditioners: a global block ILU method subregion-corrected by a block Jacobi-preconditioned GMRES method (BILU+BJ), and a global block ILU method subregion-corrected by an BoomerAMG-preconditioned GMRES method (BILU+AMG).

First we confirm that the iterative subregion correction preconditioners achieve ε_{in}-independent convergence rates, by running a suite of problems sweeping over a broad range of ε_{in}, but keeping $\varepsilon_{\text{out}} = 10^{-3}$ fixed. The outer iteration counts to achieve a tolerance of 10^{-8} are presented in Figure 5.5. As expected, BILU breaks down as ε_{in} increases, and the global problem becomes diffusion dominated. Moreover, the iterative subregion correction preconditioners converge in a fixed 4-5 iterations, confirming the predicted ε_{in}-independent convergence rates. This result provides experimental evidence suggesting Proposition 5.4.3 may still hold in the case of global preconditioners such as BILU that couple neighboring elements.

Next we analyze the performance of these preconditioners by sweeping over domains of increasing size, but fixing $\varepsilon_{\text{out}} = 10^{-2}$ and $\varepsilon_{\text{in}} = 10^{2}$ fixed for each problem. As before, we add elements to extend the domain (and outer region), but do not modify the inner or

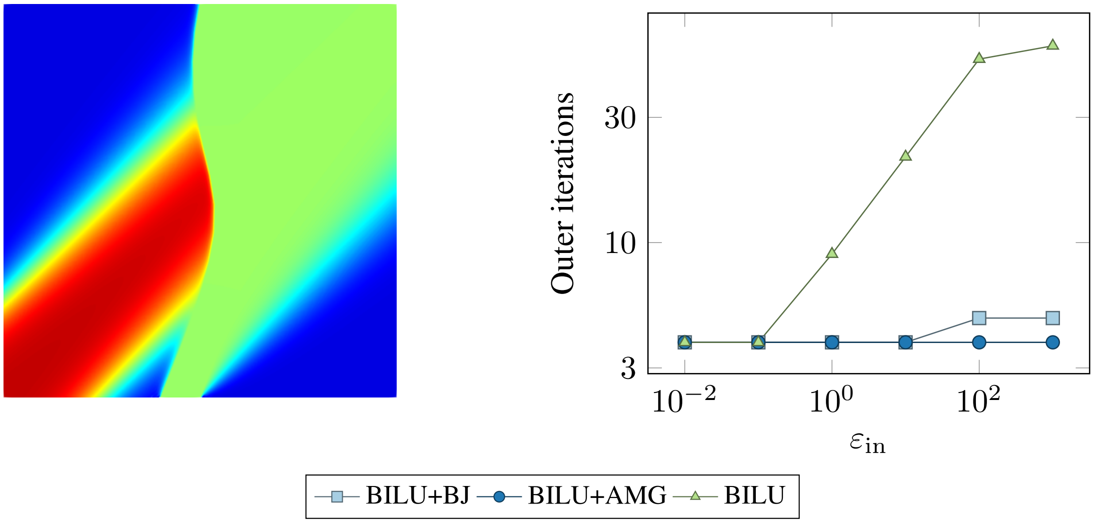


Figure 5.5: Left: Steady state solution of the variable diffusion coefficient convection-diffusion problem on a mesh containing isotropic elements. There is a thin layer of elements for which the diffusion coefficient is 10^2, a factor of 10^4 larger than the background diffusion coefficient. Right: Outer iteration counts over a sweep of ε_{in} convection-diffusion problems. The iterative subregion correction preconditioners converge in 4-5 iterations, confirming the predicted ε_{in}-independent convergence rates.

Solver configuration	n_{outer}	n_{inner}
BILU+BJ	6	355
BILU+AMG	5	37
BILU	54	—

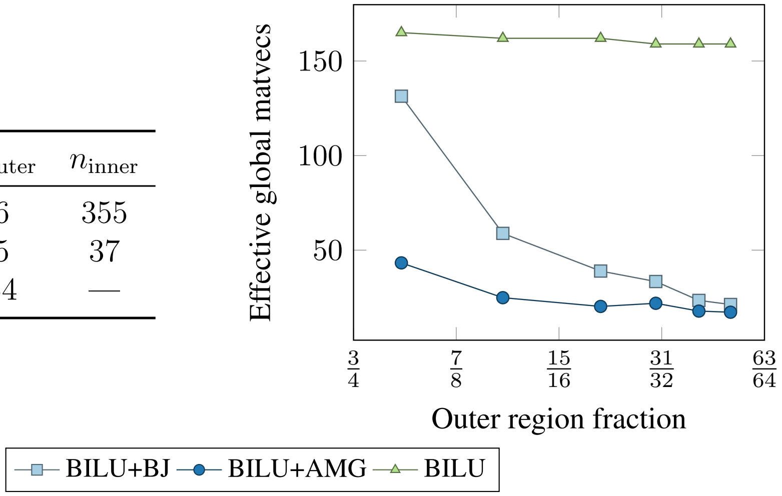


Figure 5.6: Effective global matrix-vector products required by FGMRES with three different preconditioners to converge to a tolerance of 10^{-8} for the variable diffusion coefficient convection-diffusion problem. The table presents underlying iteration counts for the $61/64 \approx 95\%$ outer region problem.

overlap regions, in order to measure the performance of our method against the outer region fraction (5.43). We solve to a tolerance of 10^{-8}, and now exclusively use the performant scaling factor from (5.10), although $\Lambda \approx 1$ for these problems because the mesh elements are mostly isotropic.

The performance model (5.42) results are shown in Figure 5.6. While BILU requires approximately 54 iterations to converge across all problems, it only takes 5-7 subregion-corrected iterations to achieve the same accuracy. Both iterative subregion correction preconditioners outperform BILU for all test problems. Again, performance savings are more substantial as the outer region grows (and so n_{sr}/n_k decreases), with a maximum 9.3× improvement reported for the 98/100 outer region fraction problem. Moreover, BILU+AMG outperforms BILU+BJ across all problems, but in particular is much more efficient when the subregion is a relatively larger fraction of the overall domain. Applying the domain knowledge that this subregion creates a locally diffusion-dominated problem, we can understand why AMG would outperform block Jacobi—this is exactly the type of problem that AMG was originally designed to solve efficiently. For outer regions taking up more than $31/32 \approx 97\%$ of the domain, the difference in performance between the two preconditioners vanishes because the impressive performance savings of the overall subregion correction method eclipse any performance savings gained from choosing a more efficient inner iterative method.

5.7 Summary

In this chapter, we have developed a new class of preconditioners for efficiently solving problems with geometrically localized stiffness. The preconditioners achieve ε-independent convergence rates, and are thus robust with respect to the subregion-localized stiffness. A cheap adaptive tolerance selection algorithm has been provided to minimize the cost of solving the subregion error equation. We have both theoretically analyzed and experimentally studied the convergence rates of our preconditioners with and without this adaptive tolerance selection.

Empirically, we reported significantly smaller outer iteration counts than naive preconditioners' across a wide array of problems. Moreover, we have highlighted in which regimes our solvers perform best: stiffer subregions whose size is relatively small compared to the global problem. On certain convection-diffusion problems, targeted iterative subregion correction preconditioners performed 10-17× faster than generic preconditioners.

Chapter 6

Case studies

In this chapter, we consider challenging case studies to illustrate some of the main properties of the methods described in the preceding chapters. We seek to demonstrate the application of these techniques to large eddy simulation (LES) of turbulent flow problems modeled by the Navier-Stokes equations. The incompressible flow problem will be solved utilizing the matrix-free solvers developed in Chapter 4. The compressible flow problems possess geometrically localized stiffness in the boundary layer, and therefore will be solved utilizing the iterative subregion correction preconditioners developed in Chapter 5.

6.1 Incompressible flow: Taylor-Green vortex

We first consider the incompressible Taylor-Green vortex, which is a standard benchmark case often used to assess the accuracy of high-order methods [18, 17]. This problem represents a simple model for the development of turbulence and resulting cascade of energy from large to small scales [122]. We use the problem configuration as defined in the first international workshop on high-order CFD methods [132].

We solve the 3D incompressible Navier-Stokes equations (2.10) with our matrix-free operators over the fully periodic cube $\Omega = [-\pi, \pi]^3$. The initial state is set to

$$\begin{aligned}
u_1(x,y,z) &= \sin(x)\cos(y)\sin(z),\\
u_2(x,y,z) &= -\cos(x)\sin(y)\sin(z),\\
u_3(x,y,z) &= 0,\\
p(x,y,z) &= p_0 + \frac{p_0\left\|\boldsymbol{u_0}\right\|_2^2}{16RT_0}\left(\cos(2x)+\cos(2y)\right)\left(\cos(2z)+2\right).
\end{aligned}$$

The Reynolds number is chosen to be Re $= 1600$, initial temperature T_0 is assumed uniform, and R is the universal gas constant. The equations are integrated with BDF3 until a final time of $t = 20$ using a time step of $\Delta t = 2.5 \times 10^{-3}$. We consider the following three mesh configurations:

- $p = 3$, $24 \times 24 \times 24$ grid, 439,276 DoFs per component.
- $p = 7$, $12 \times 12 \times 12$ grid, 650,701 DoFs per component.
- $p = 11$, $6 \times 6 \times 6$ grid, 346,969 DoFs per component.

In Figure 6.1, we display the time evolution of Q-criterion isosurfaces, colored by velocity magnitude. The quantity Q is defined by

$$Q = \frac{1}{2} \sum_{i,j=1}^{3} \frac{\partial u_j}{\partial x_i} \frac{\partial u_i}{\partial x_j}, \tag{6.1}$$

and is commonly used for vortex identification [67, 48]. These isosurfaces clearly display the evolution from smooth, large-scale structures to small-scale turbulent structures.

We compare the results obtained using the present high-order finite element flow solver with reference data obtained using a de-aliased pseudo-spectral method [109]. The quantities of interest for this comparison are the total kinetic energy

$$E_k = \frac{1}{|\Omega|} \int_\Omega \frac{\boldsymbol{u}^T \boldsymbol{u}}{2} d\boldsymbol{x} \tag{6.2}$$

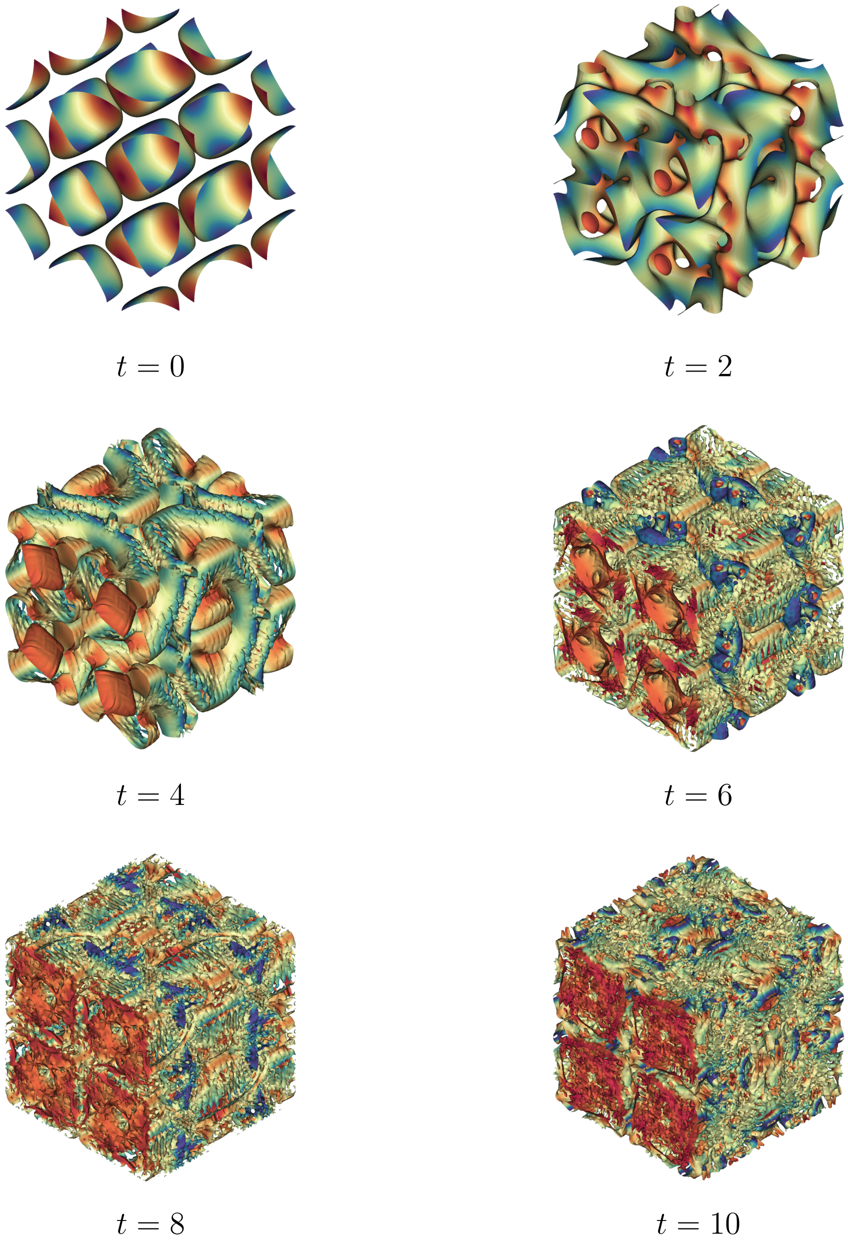
and the kinetic energy dissipation rate

$$\epsilon = -\frac{dE_k}{dt}. \tag{6.3}$$

The time evolution of these quantities is shown in Figure 6.2. All of the configurations considered result in good agreement with the reference data. The lowest order case (polynomial degree $p = 3$) is the most dissipative, slightly under-predicting the total kinetic energy after about $t = 10$. The highest order case we considered (polynomial degree $p = 11$) gives results of comparable accuracy to the $p = 7$ case, with roughly half as many degrees of freedom.

6.2 High Reynolds number compressible flow

Next we explore the performance of our iterative subregion correction preconditioners on a fluid flow problem with mesh-dependent geometrically localized stiffness. Specifically, we solve the 2D compressible Navier-Stokes equations (2.2), at Re $\approx 2.25 \times 10^6$ around the NACA 0012 airfoil. $M_\infty = 0.25$ is the far-field Mach number. The chord length is normalized to be 1, and unstructured quadrilateral meshes are generated for the domains $[-R, 2R] \times [-R, R]$, with $R \in \{5, 10, 20, 30, 40\}$. For each mesh, the leading edge of the airfoil is placed at the origin. Thus, all domain boundaries are at least 5 chord lengths from the airfoil, and the wake region has a minimum length of 10 chord lengths. Far-field conditions



$t = 0$

$t = 2$

$t = 4$

$t = 6$

$t = 8$

$t = 10$

Figure 6.1: Time evolution of the incompressible Taylor-Green vortex, showing $Q = 0.1$ isosurfaces colored by velocity magnitude.

are enforced at domain boundaries, and a no-slip wall condition is enforced at the surface of the airfoil.

We run a wall-resolved implicit large eddy simulation (WRILES), which uses natural dissipation from the high-order ($p = 3$) DG method as a subgrid closure model [130, 52]. Resolving

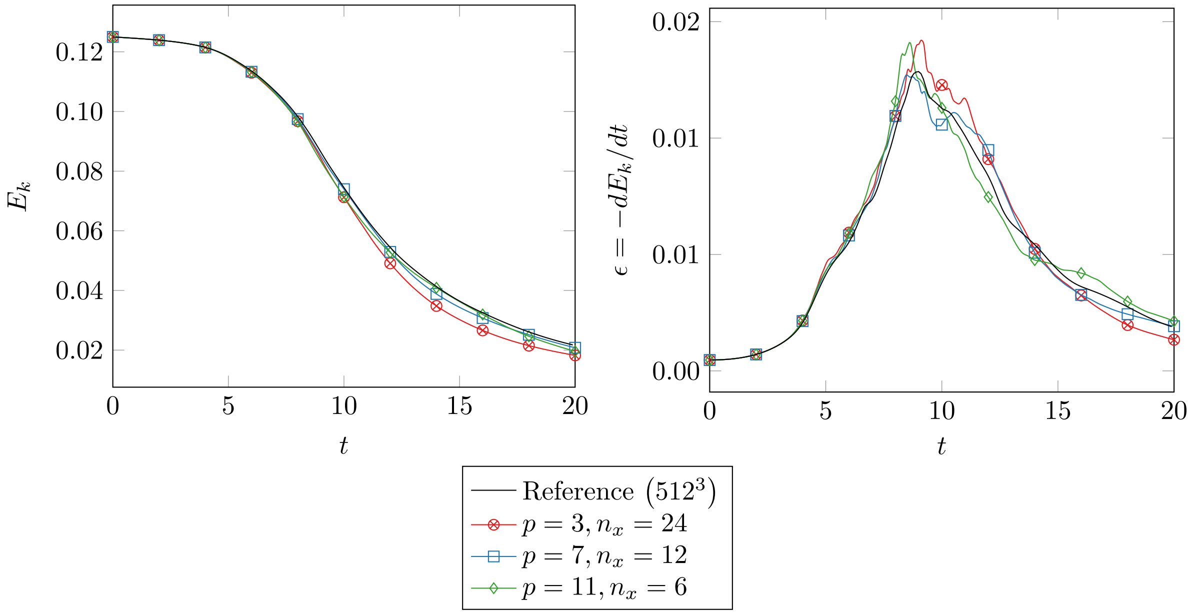


Figure 6.2: Time evolution of total kinetic energy and kinetic energy dissipation rate for the incompressible Taylor-Green vortex. Comparison with reference data from a fully-resolved pseudo-spectral method with 512^3 degrees of freedom per component.

the boundary layer near the wall requires $y^+ < 1$ for the entire length of the boundary layer. With such a high Reynolds number, $y^+ = 1$ corresponds to a distance of about 4×10^{-6} units from the surface of the airfoil. Therefore, we perform 12 structured refinements in the transverse direction around the airfoil, resulting in highly anisotropic elements in the boundary layer, with the largest element aspect ratio approximately 4096:1. Our mesh is shown in Figure 6.3. This refinement causes a geometrically localized stiffness in the boundary layer. So we choose our subregion to be exactly those elements that are created by the refinement. This choice of 778 elements for the subregion naturally contains overlap elements as well—the mesh-induced ε of elements with an aspect ratio of 2:1 furthest from the airfoil are orders of magnitude smaller than the ε of elements on the wall.

We repeat our experiment from Section 5.6.1, testing the performance of different instances from our class of iterative subregion correction preconditioners for this problem. Specifically, we solve the linearized problem (2.53) around a fully formed flow at $t = 4$ with $\Delta t = 10^{-3}$ for the five different domains defined by $R \in \{5, 10, 20, 30, 40\}$ to a relative tolerance of 10^{-8}. We test FGMRES preconditioned by block Jacobi (BJ), block ILU (BILU), a global block Jacobi method subregion-corrected by a block Jacobi-preconditioned GMRES method (BJ+BJ), and a global block Jacobi method subregion-corrected by a block ILU-preconditioned GMRES method (BJ+BILU). The performance model (5.42) results are shown in Figure 6.4.

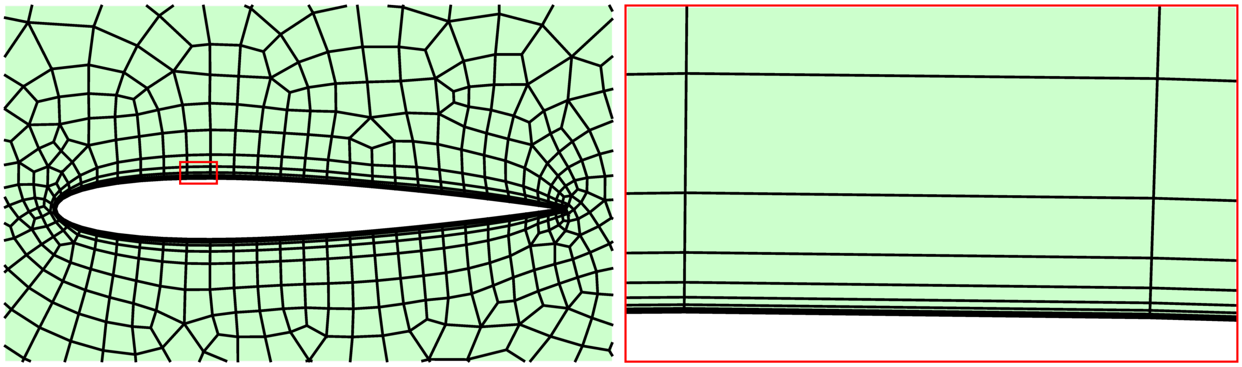


Figure 6.3: Two-dimensional mesh used for the Re $= 2.25 \times 10^6$ flow over the NACA 0012 airfoil. The right hand side zoom-in shows elements with aspect ratios greater than 4000:1.

Solver configuration	n_{outer}	n_{inner}
BJ+BJ	25	374
BJ+BILU	16	21
BJ	273	—
BILU	18	—

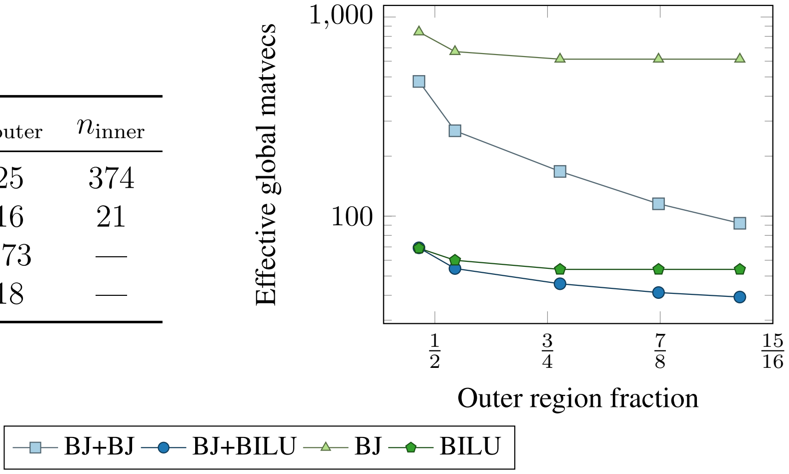


Figure 6.4: Effective global matrix-vector products required by FGMRES with four different preconditioners to converge to a tolerance of 10^{-8} for the unsteady Navier-Stokes problem with Re $= 2.25 \times 10^6$ and $\Delta t = 10^{-3}$. The table presents underlying iteration counts for the 7/8 outer region problem.

As before, we see that the performance savings of the iterative subregion correction preconditioners improve as the outer region increases, with a 5.3× and 14.8× improvement over the block Jacobi method reported for the 7/8 outer region fraction problem. In the limit as the cost of the inner iterations at the subregion vanishes relative to the cost of the outer iterations, the horizontal asymptote for this problem will occur at 36 effective global matvecs, which would be a 17× improvement over pure block Jacobi and 1.5× improvement over block ILU.

These results corroborate previous findings that block ILU is an effective preconditioner for

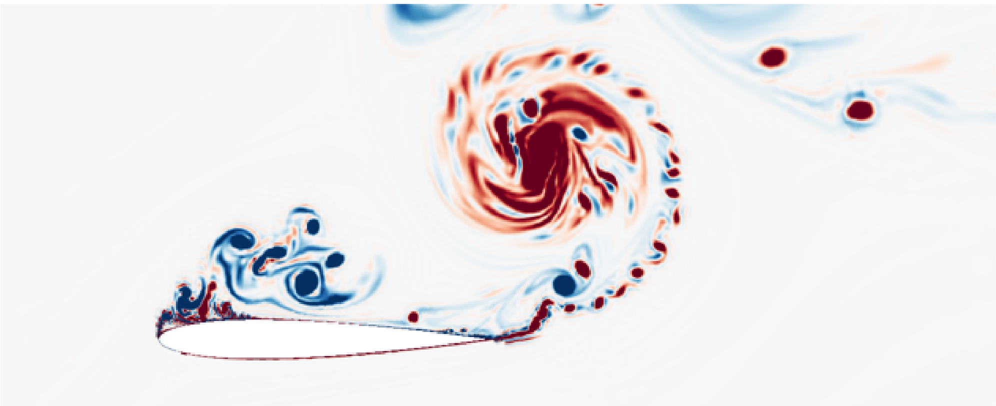


Figure 6.5: Density plot around the NACA 0012 airfoil, showing massive separation of the flow at $\mathrm{Re} = 10^6$ and a 30° angle of attack.

this problem [105], but furthermore highlight that the reason for this efficiency is found in the boundary layer elements. Indeed, block ILU converges in one tenth as many inner iterations as block Jacobi. By element-wise decoupling the system, block Jacobi fails to effectively precondition the boundary layer elements in both the global solver and subregion solver. However, Figure 6.4 shows that block Jacobi is a cheap and efficient preconditioner in the far-field elements, and so the block Jacobi method subregion-corrected by an inner block ILU preconditioner is the most efficient preconditioner across all tested domain sizes.

We note that that our BJ+BJ iterative subregion correction preconditioner is fully matrix-free, allowing it to be used effectively for matrix-free methods we developed in Chapter 3. While its performance is not as impressive as the BJ+BILU solver for relatively smaller outer region fractions, it still converges to the 17× improvement over pure block Jacobi as the outer region fraction converges to 1.

6.3 Compressible flow with massive separation

Next, we explore performance of our iterative subregion correction preconditioners on a practical example: LES of massively separated turbulent flow. Specifically, we again solve the 2D compressible Navier-Stokes equations (2.2) over the NACA 0012 airfoil with $\mathrm{Re} = 10^6$, and a 30° angle of attack. We assume isentropic flow, so that entropy satisfies (2.8), and we can model one fewer component. The chord length is normalized to be 1, and an unstructured quadrilateral mesh is generated for the domain $[-10, 20] \times [-10, 10]$. The leading edge of the airfoil is placed at the origin. The mesh is structurally refined 7 times in the transverse direction around the airfoil to adequately mesh the boundary layer, with the largest element aspect ratio approximately $128 : 1$. To a lesser extent, the mesh is refined in the wake region where the eddies separate. The density of the flow is shown in Figure 6.5.

Solver configuration	n_{outer}	n_{inner}
BJ+BILU	29	8
BJ	78	—

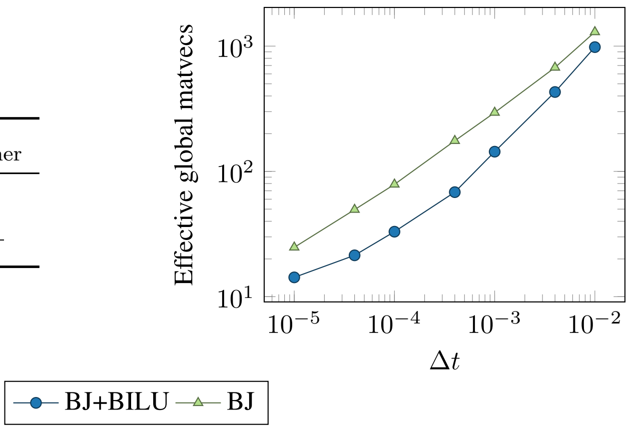


Figure 6.6: Effective global matrix-vector products required by BJ and BJ+BILU to solve the linearized LES of massively separated flow problem to a tolerance of 10^{-8}. The iterative subregion correction preconditioner costs 25%-60% less than block Jacobi across all Δt. The table presents underlying iteration counts for the $\Delta t = 4 \times 10^4$ problem.

As shown by the previous example, the boundary layer elements of this WRILES create a stiffer problem than the rest of the mesh. Therefore, we choose our subregion to be those 3102 elements created by the refinement, resulting in 7/10 of the elements belonging to the outer region. Moreover, the previous example highlighted how BJ+BILU was the most performant solver for a similar flow problem. So on this massively separated flow problem, we explore the robustness of this solver with respect to nondimensionalized time step Δt. Figure 6.6 shows the performance results.

First of all we note that the iterative subregion correction preconditioner costs less than block Jacobi across all Δt in this experiment. Also, savings are more substantial (up to 60%) for $4 \times 10^{-5} \leq \Delta t \leq 10^{-3}$, a range of Δt with which practical time-implicit simulations are typically run [99, 105, 118]. As $\Delta t \to 0$, the problem approaches the explicit regime and block Jacobi already performs well across the entire domain, requiring less than 10 iterations to converge. There is little room for performance savings at this extreme with the boundary layer problem so simple that a naive block Jacobi method already performs well. Moreover, as $\Delta t \to 1$, the stiffness in the outer region increases, so that block Jacobi requires more than 200 iterations to converge. No matter the subregion solver performance, an iterative subregion correction preconditioner does not improve convergence in the outer region, and so relatively less impressive performance savings are expected. If the boundary layer was more refined, thereby increasing the geometrically localized stiffness, savings would be larger. Regardless, across the broad spectrum of stiffness induced by different Δt, the iterative subregion correction preconditioner provides moderate computational savings.

Chapter 7

Conclusions

High-order methods continue to be a promising area of research, particularly in the field of computational fluid dynamics. High-order discontinuous Galerkin methods in particular have demonstrated highly accurate solutions with less degrees of freedom than low-order alternative methods. Moreover, implementations of these methods on modern high performance computing architectures have been performant at scale, with low memory and storage costs.

The matrix-free methods we presented utilize sum-factorization techniques to achieve optimal memory usage and operation counts. Optimized GPU kernels obtained a throughput of over a billion degrees of freedom per second on a single Nvidia V100 GPU. Moreover, we showcased the computational benefits of high-order simulations on the GPU, as our optimized GPU kernels outperformed the 20-core CPU implementation by more than a factor of 10.

We developed a suite of matrix-free linear solvers for incompressible flow. Our mass, Poisson, and Helmholtz sub-problem solvers do not require the costly assembly of high-order system matrices, and instead achieve optimal linear scaling of memory and operation counts with the number of degrees of freedom. The preconditioners we developed were shown to be robust with respect to mesh size, polynomial degree, and time step on a range of two and three dimensional test problems.

We also developed iterative subregion correction preconditioners for different problems exhibiting geometrically localized stiffness. We proved and empirically showed that this class of preconditioners is robust with respect to the subregion-localized stiffness. Moreover, these preconditioners perform relatively better as the subregion-localized stiffness and outer region fraction increase. The flexibility of choosing effective sub-preconditioners allows one to leverage domain-specific knowledge of a given problem to create a robust and performant iterative subregion correction preconditioner. On certain convection-diffusion problems, targeted iterative subregion correction preconditioners performed 10-17× faster than generic preconditioners. On the massively separated flow problem, which is more challenging for our approach, we still obtained up to 60% performance savings.

Future work can be done on several of these topics. Firstly, optimizing the matrix-free preconditioners so that they too can be used efficiently on the GPU is an important step towards removing unnecessary memory transfers to and from the device during an iterative method. While the methods we have developed are optimal in memory usage, the memory access patterns are currently not well suited for GPU-accelerated architectures, leaving room for improvement. Likewise, parallelization of the iterative subregion correction preconditioners should be further explored. The subregion iterations may dominate overall performance, but the subregion may be much smaller than the overall domain, so robust parallelization of the subregion work is paramount to scalable simulations. Finally, because the performance of these preconditioners depends so heavily upon the size of the subregion, optimal subregion selection algorithms should be explored.

Bibliography

[1] Roger Alexander. "Diagonally implicit Runge–Kutta methods for stiff ODE's". In: *SIAM Journal on Numerical Analysis* 14.6 (1977), pp. 1006–1021. DOI: 10.1137/0714068.

[2] Robert Anderson et al. "MFEM: A modular finite element methods library". In: *Computers & Mathematics with Applications* 81 (2021), pp. 42–74. DOI: 10.1016/j.camwa.2020.06.009.

[3] Paola F Antonietti and Paul Houston. "A class of domain decomposition preconditioners for hp-discontinuous Galerkin finite element methods". In: *Journal of Scientific Computing* 46.1 (2011), pp. 124–149. DOI: 10.1007/s10915-010-9390-1.

[4] Douglas N. Arnold et al. "Unified analysis of discontinuous Galerkin methods for elliptic problems". In: *SIAM Journal on Numerical Analysis* 39.5 (2002), pp. 1749–1779. DOI: 10.1137/S0036142901384162.

[5] KJ Badcock, BE Richards, and MA Woodgate. "Elements of computational fluid dynamics on block structured grids using implicit solvers". In: *Progress in Aerospace Sciences* 36.5-6 (2000), pp. 351–392. DOI: 10.1016/S0376-0421(00)00005-1.

[6] Jonás D. De Basabe, Mrinal K. Sen, and Mary F. Wheeler. "Seismic wave propagation in fractured media: A discontinuous Galerkin approach". In: *SEG Technical Program Expanded Abstracts 2011*. 2012, pp. 2920–2924. DOI: 10.1190/1.3627801.

[7] F Bassi and S Rebay. "A high order discontinuous Galerkin method for compressible turbulent flows". In: *Discontinuous Galerkin Methods*. Springer, 2000, pp. 77–88. DOI: 10.1007/978-3-642-59721-3_4.

[8] F Bassi and S Rebay. "GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations". In: *Discontinuous Galerkin Methods*. Springer, 2000, pp. 197–208. DOI: 10.1007/978-3-642-59721-3_14.

[9] Francesco Bassi and Stefano Rebay. "A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations". In: *Journal of computational physics* 131.2 (1997), pp. 267–279. DOI: 10.1006/jcph.1996.5572.

[10] Francesco Bassi et al. “A high-order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows”. In: *Proceedings of the 2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics*. Antwerpen, Belgium. 1997, pp. 99–109.

[11] Peter Bastian et al. “Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations”. In: *Journal of Computational Physics* 394 (Oct. 2019), pp. 417–439. DOI: 10.1016/j.jcp.2019.06.001.

[12] John B Bell, Phillip Colella, and Harland M Glaz. “A second-order projection method for the incompressible Navier-Stokes equations”. In: *Journal of Computational Physics* 85.2 (Dec. 1989), pp. 257–283. DOI: 10.1016/0021-9991(89)90151-4.

[13] Pedro D. Bello-Maldonado and Paul F. Fischer. “Scalable low-order finite element preconditioners for high-order spectral element Poisson solvers”. In: *SIAM Journal on Scientific Computing* 41.5 (2019), S2–S18. DOI: 10.1137/18M1194997.

[14] Michele Benzi, Gene H. Golub, and Jörg Liesen. “Numerical solution of saddle point problems”. In: *Acta Numerica* 14 (Apr. 2005), pp. 1–137. DOI: 10.1017/s0962492904000212.

[15] Julien Bodart, Johan Larsson, and Parviz Moin. “Large eddy simulation of high-lift devices”. In: *21st AIAA Computational Fluid Dynamics Conference*. 2013. DOI: 10.2514/6.2013-2724.

[16] Sanjeeb T Bose and George Ilhwan Park. “Wall-modeled large-eddy simulation for complex turbulent flows”. In: *Annual review of fluid mechanics* 50 (2018), pp. 535–561. DOI: 10.1146/annurev-fluid-122316-045241.

[17] M. E. Brachet et al. “The Taylor-Green vortex and fully developed turbulence”. In: *Journal of Statistical Physics* 34.5-6 (Mar. 1984), pp. 1049–1063. DOI: 10.1007/bf01009458.

[18] Marc E. Brachet et al. “Small-scale structure of the Taylor-Green vortex”. In: *Journal of Fluid Mechanics* 130.-1 (May 1983), p. 411. DOI: 10.1017/s0022112083001159.

[19] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. Society for Industrial and Applied Mathematics, Jan. 1995. DOI: 10.1137/1.9781611971224.

[20] F. Brezzi et al. “Discontinuous Galerkin approximations for elliptic problems”. In: *Numerical Methods for Partial Differential Equations* 16.4 (2000), pp. 365–378. DOI: 10.1002/1098-2426(200007)16:4<365::AID-NUM2>3.0.CO;2-Y.

[21] Franco Brezzi and Richard S Falk. “Stability of higher-order Hood–Taylor methods”. In: *SIAM Journal on Numerical Analysis* 28.3 (1991), pp. 581–590. DOI: 10.1137/0728032.

[22] William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000. ISBN: 978-0-89871-950-5. DOI: 10.1137/1.9780898719505.

[23] Kolja Brix, Claudio Canuto, and Wolfgang Dahmen. "Nested dyadic grids associated with Legendre-Gauss-Lobatto grids". In: *Numerische Mathematik* 131.2 (Dec. 2014), pp. 205–239. DOI: 10.1007/s00211-014-0691-4.

[24] David L. Brown, Ricardo Cortez, and Michael L. Minion. "Accurate projection methods for the incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 168.2 (Apr. 2001), pp. 464–499. DOI: 10.1006/jcph.2001.6715.

[25] Jed Brown et al. *CEED ECP Milestone Report: Public release of CEED 1.0*. Tech. rep. WBS 2.2.6.06, CEED-MS13. U.S. Department of Energy, Mar. 2018. DOI: 10.5281/zenodo.2542343.

[26] Tan Bui-Thanh and Omar Ghattas. "Analysis of an hp-nonconforming discontinuous Galerkin spectral element method for wave propagation". In: *SIAM Journal on Numerical Analysis* 50.3 (2012), pp. 1801–1826. DOI: 10.1137/110828010.

[27] Graham V Candler, Pramod K Subbareddy, and Joseph M Brock. "Advances in computational fluid dynamics methods for hypersonic flows". In: *Journal of Spacecraft and Rockets* 52.1 (2015), pp. 17–28. DOI: 10.2514/1.A33023.

[28] Claudio Canuto. "Stabilization of spectral methods by finite element bubble functions". In: *Computer Methods in Applied Mechanics and Engineering* 116.1-4 (Jan. 1994), pp. 13–26. DOI: 10.1016/s0045-7825(94)80004-9.

[29] Claudio Canuto, Paola Gervasio, and Alfio Quarteroni. "Finite-element preconditioning of G-NI spectral methods". In: *SIAM Journal on Scientific Computing* 31.6 (Jan. 2010), pp. 4422–4451. DOI: 10.1137/090746367.

[30] Claudio Canuto and Alfio Quarteroni. "Preconditioned minimal residual methods for Chebyshev spectral calculations". In: *Journal of Computational Physics* 60.2 (Sept. 1985), pp. 315–337. DOI: 10.1016/0021-9991(85)90010-5.

[31] Claudio Canuto et al. *Spectral methods. Evolution to complex geometries and applications to fluid dynamics*. Springer Berlin Heidelberg, 2007. DOI: 10.1007/978-3-540-30728-0.

[32] Jakub Cerveny, Veselin Dobrev, and Tzanio Kolev. "Nonconforming mesh refinement for high-order finite elements". In: *SIAM Journal on Scientific Computing* 41.4 (2019), pp. C367–C392. DOI: 10.1137/18M1193992.

[33] Alexandre Joel Chorin. "A numerical method for solving incompressible viscous flow problems". In: *Journal of Computational Physics* 2.1 (Aug. 1967), pp. 12–26. DOI: 10.1016/0021-9991(67)90037-x.

[34] Alexandre Joel Chorin. "Numerical solution of the Navier-Stokes equations". In: *Mathematics of Computation* 22.104 (1968), pp. 745–745. DOI: 10.1090/s0025-5718-1968-0242392-2.

[35] Bernardo Cockburn, Suchung Hou, and Chi-Wang Shu. "The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV. The multidimensional case". In: *Mathematics of Computation* 54.190 (1990), pp. 545–581. DOI: 10.1090/S0025-5718-1990-1010597-0.

[36] Bernardo Cockburn, George E Karniadakis, and Chi-Wang Shu. "The development of discontinuous Galerkin methods". In: *Discontinuous Galerkin Methods*. Springer, 2000, pp. 3–50. DOI: 10.1007/978-3-642-59721-3_1.

[37] Bernardo Cockburn, San-Yih Lin, and Chi-Wang Shu. "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One-dimensional systems". In: *Journal of Computational Physics* 84.1 (1989), pp. 90–113. DOI: 10.1016/0021-9991(89)90183-6.

[38] Bernardo Cockburn and Chi-Wang Shu. "Runge-Kutta discontinuous Galerkin methods for convection-dominated problems". In: *Journal of scientific computing* 16.3 (2001), pp. 173–261. DOI: 10.1023/A:1012873910884.

[39] Bernardo Cockburn and Chi-Wang Shu. "The local discontinuous Galerkin method for time-dependent convection-diffusion systems". In: *SIAM Journal on Numerical Analysis* 35.6 (1998), pp. 2440–2463. DOI: 10.1137/S0036142997316712.

[40] Bernardo Cockburn and Chi-Wang Shu. "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II. General framework". In: *Mathematics of computation* 52.186 (1989), pp. 411–435. DOI: 10.1090/S0025-5718-1989-0983311-4.

[41] R. Courant, K. Friedrichs, and H. Lewy. "Über die partiellen Differenzengleichungen der mathematischen Physik". In: *Mathematische Annalen* 100 (1928), pp. 32–74. DOI: 10.1007/BF01448839.

[42] Charles Francis Curtiss and Joseph O Hirschfelder. "Integration of stiff equations". In: *Proceedings of the National Academy of Sciences of the United States of America* 38.3 (1952), p. 235. DOI: 10.1073/pnas.38.3.235.

[43] Eduardo F D'Azevedo, Peter A Forsyth, and Wei-Pai Tang. "Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems". In: *SIAM Journal on Matrix Analysis and Applications* 13.3 (1992), pp. 944–961. DOI: 10.1137/0613057.

[44] M. O. Deville and E. H. Mund. "Finite-element preconditioning for pseudospectral solutions of elliptic problems". In: *SIAM Journal on Scientific and Statistical Computing* 11.2 (Mar. 1990), pp. 311–342. DOI: 10.1137/0911019.

[45] Michel O Deville, Paul F Fischer, EH Mund, et al. *High-order methods for incompressible fluid flow*. Vol. 9. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511546792.

[46] Veselin Dobrev et al. *CEED ECP Milestone Report: Identify initial kernels, bake-off problems (benchmarks) and miniapps*. Tech. rep. WBS 1.2.5.3.04, CEED-MS6. U.S. Department of Energy, 2017. DOI: 10.5281/zenodo.2542333.

[47] Jim Douglas and Todd Dupont. "Interior penalty procedures for elliptic and parabolic Galerkin methods". In: *Computing methods in applied sciences*. Springer, 1976, pp. 207–216. DOI: 10.1007/BFb0120591.

[48] Yves Dubief and Franck Delcayre. "On coherent-vortex identification in turbulence". In: *Journal of Turbulence* 1 (Jan. 2000), N11. DOI: 10.1088/1468-5248/1/1/011.

[49] Richard E. Ewing et al. "Preconditioning indefinite systems arising from mixed finite element discretization of second-order elliptic problems". In: *Preconditioned Conjugate Gradient Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 28–43. DOI: 10.1007/BFb0090900.

[50] Robert D Falgout and Ulrike Meier Yang. "hypre: A library of high performance preconditioners". In: *International Conference on Computational Science*. Springer. 2002, pp. 632–641. DOI: 10.1007/3-540-47789-6_66.

[51] Niklas Fehn, Wolfgang A. Wall, and Martin Kronbichler. "A matrix-free high-order discontinuous Galerkin compressible Navier-Stokes solver: A performance comparison of compressible and incompressible formulations for turbulent incompressible flows". In: *International Journal for Numerical Methods in Fluids* 89.3 (Oct. 2018), pp. 71–102. DOI: 10.1002/fld.4683.

[52] Pablo Fernandez, Ngoc-Cuong Nguyen, and Jaime Peraire. *On the ability of discontinuous Galerkin methods to simulate under-resolved turbulent flows*. 2018. DOI: 10.48550/ARXIV.1810.09435.

[53] Krzysztof J Fidkowski et al. "p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations". In: *Journal of Computational Physics* 207.1 (2005), pp. 92–113. DOI: 10.1016/j.jcp.2005.01.005.

[54] Paul F Fischer. "An overlapping Schwarz method for spectral element solution of the incompressible Navier–Stokes equations". In: *Journal of Computational Physics* 133.1 (1997), pp. 84–101. DOI: 10.1006/jcph.1997.5651.

[55] Paul F Fischer and James W Lottes. "Hybrid Schwarz-multigrid methods for the spectral element method: Extensions to Navier-Stokes". In: *Domain Decomposition Methods in Science and Engineering*. Springer, 2005, pp. 35–49. DOI: 10.1007/3-540-26825-1_3.

[56] Matteo Franciolini, Andrea Crivellini, and Alessandra Nigro. "On the efficiency of a matrix-free linearly implicit time integration strategy for high-order discontinuous Galerkin solutions of incompressible turbulent flows". In: *Computers & Fluids* 159 (Dec. 2017), pp. 276–294. DOI: 10.1016/j.compfluid.2017.10.008.

[57] Burkely T Gallo et al. "Breaking new ground in severe weather prediction: The 2015 NOAA/hazardous weather testbed spring forecasting experiment". In: *Weather and Forecasting* 32.4 (2017), pp. 1541–1568. DOI: 10.1175/WAF-D-16-0178.1.

[58] Gregor Gassner and David A Kopriva. "A comparison of the dispersion and dissipation errors of Gauss and Gauss–Lobatto discontinuous Galerkin spectral element methods". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2560–2579. DOI: 10.1137/100807211.

[59] Sigal Gottlieb and Chi-Wang Shu. "Total variation diminishing Runge-Kutta schemes". In: *Mathematics of computation* 67.221 (1998), pp. 73–85. DOI: 10.1090/S0025-5718-98-00913-2.

[60] Michael Griebel and Peter Oswald. "On the abstract theory of additive and multiplicative Schwarz algorithms". In: *Numerische Mathematik* 70.2 (1995), pp. 163–180. DOI: 10.1007/s002110050115.

[61] Jean-Luc Guermond, Peter D. Minev, and Jianhua Shen. "An overview of projection methods for incompressible flows". In: *Computer Methods in Applied Mechanics and Engineering* 195 (2006), pp. 6011–6045. DOI: 10.1016/j.cma.2005.10.010.

[62] Ernst Hairer and Gerhard Wanner. *Solving ordinary differential equations II. Stiff and differential-algebraic problems*. Springer Berlin Heidelberg, 1996. DOI: 10.1007/978-3-642-05221-7.

[63] Magnus R Hestenes and Eduard Stiefel. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49.6 (1952), p. 409. DOI: 10.6028/jres.049.044.

[64] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007. ISBN: 9780387720678. DOI: 10.1007/978-0-387-72067-8.

[65] Mark Hoemmen. *Communication-avoiding Krylov subspace methods*. University of California, Berkeley, 2010. URL: https://www.proquest.com/docview/749357839.

[66] Chang-Ho Hong and Inderjit Chopra. "Aeroelastic stability analysis of a composite rotor blade". In: *Journal of the American Helicopter Society* 30.2 (1985), pp. 57–67. DOI: 10.4050/JAHS.30.57.

[67] Julian CR Hunt, Alan A Wray, and Parviz Moin. "Eddies, streams, and convergence zones in turbulent flows". In: *Studying Turbulence Using Numerical Simulation Databases, 2. Proceedings of the 1988 Summer Program* (Dec. 1988). URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19890015184.pdf.

[68] Arieh Iserles. *A first course in the numerical analysis of differential equations*. 44. Cambridge university press, 1996. ISBN: 9780511995569. DOI: 10.1017/CBO9780511995569.

[69] Volker John, Gunar Matthies, and Joachim Rang. "A comparison of time-discretization/linearization approaches for the incompressible Navier-Stokes equations". In: *Computer Methods in Applied Mechanics and Engineering* 195.44-47 (Sept. 2006), pp. 5995–6010. DOI: 10.1016/j.cma.2005.10.007.

[70] Samuel Kanner and Per-Olof Persson. "Validation of a high-order large-eddy simulation solver using a vertical-axis wind turbine". In: *AIAA Journal* 54.1 (2016), pp. 101–112. DOI: 10.2514/1.j054138.

[71] George Em Karniadakis, Moshe Israeli, and Steven A Orszag. "High-order splitting methods for the incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 97.2 (Dec. 1991), pp. 414–443. DOI: 10.1016/0021-9991(91)90007-8.

[72] Carl T Kelley. *Iterative methods for linear and nonlinear equations*. SIAM, 1995. DOI: 10.1137/1.9781611970944.

[73] A. Klöckner, T. Warburton, and J. S. Hesthaven. "Viscous shock capturing in a time-explicit discontinuous Galerkin method". In: *Mathematical Modelling of Natural Phenomena* 6.3 (2011), pp. 57–83. DOI: 10.1051/mmnp/20116303.

[74] A. Klöckner et al. "Nodal discontinuous Galerkin methods on graphics processors". In: *Journal of Computational Physics* 228.21 (Nov. 2009), pp. 7863–7882. DOI: 10.1016/j.jcp.2009.06.041.

[75] Dana A Knoll and William J Rider. "A multigrid preconditioned Newton–Krylov method". In: *SIAM Journal on Scientific Computing* 21.2 (1999), pp. 691–710. DOI: 10.1137/S1064827598332709.

[76] L. I. G. Kovasznay. "Laminar flow behind a two-dimensional grid". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 44.1 (1948), pp. 58–62. DOI: 10.1017/S0305004100023999.

[77] Martin Kronbichler and Katharina Kormann. "A generic interface for parallel cell-based finite element operator application". In: *Computers & Fluids* 63 (2012), pp. 135–147. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2012.04.012.

[78] Martin Kronbichler and Karl Ljungkvist. "Multigrid for matrix-free high-order finite element computations on graphics processors". In: *ACM Transactions on Parallel Computing* 6.1 (May 2019), pp. 1–32. DOI: 10.1145/3322813.

[79] Andrea S Les et al. "Quantification of hemodynamics in abdominal aortic aneurysms during rest and exercise using magnetic resonance imaging and computational fluid dynamics". In: *Annals of biomedical engineering* 38.4 (2010), pp. 1288–1313. DOI: 10.1007/s10439-010-9949-x.

[80] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007. DOI: 10.1137/1.9780898717839.

[81] Randall J LeVeque and D.G. Crighton. *Finite volume methods for hyperbolic problems*. Cambridge university press, 2002. ISBN: 9780511791253. DOI: 10.1017/CBO9780511791253.

[82] Randall J. LeVeque, David L. George, and Marsha J. Berger. "Tsunami modelling with adaptively refined finite volume methods". In: *Acta Numerica* 20 (2011), pp. 211–289. DOI: 10.1017/S0962492911000043.

[83] James W Lottes and Paul F Fischer. “Hybrid multigrid/Schwarz algorithms for the spectral element method”. In: *Journal of Scientific Computing* 24.1 (2005), pp. 45–78. DOI: 10.1007/s10915-004-4787-3.

[84] Yu Lv and Matthias Ihme. “Discontinuous Galerkin method for multicomponent chemically reacting flows and combustion”. In: *Journal of Computational Physics* 270 (2014), pp. 105–137. DOI: 10.1016/j.jcp.2014.03.029.

[85] Thomas A Manteuffel, John Ruge, and Ben S Southworth. “Nonsymmetric algebraic multigrid based on local approximate ideal restriction (ℓAIR)”. In: *SIAM Journal on Scientific Computing* 40.6 (2018), A4105–A4130. DOI: 10.1137/17M1144350.

[86] *MFEM: Modular Finite Element Methods Library*. https://mfem.org. DOI: 10.11578/dc.20171025.1248.

[87] Raouf Mobasheri, Zhijun Peng, and Seyed Mostafa Mirsalim. “Analysis the effect of advanced injection strategies on engine performance and pollutant emissions in a heavy duty DI-diesel engine by CFD modeling”. In: *International Journal of Heat and Fluid Flow* 33.1 (2012), pp. 59–69. DOI: 10.1016/j.ijheatfluidflow.2011.10.004.

[88] Michael F Modest and Sandip Mazumder. *Radiative heat transfer*. Elsevier Science, 2013. ISBN: 978-0-12-386944-9. DOI: 10.1016/C2010-0-65874-3.

[89] Marghoob Mohiyuddin et al. “Minimizing communication in sparse matrix solvers”. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. IEEE. 2009, pp. 1–12. DOI: 10.1145/1654059.1654096.

[90] Ramachandran D. Nair, Michael N. Levy, and Peter H. Lauritzen. “Emerging Numerical Methods for Atmospheric Modeling”. In: *Numerical Techniques for Global Atmospheric Models*. Ed. by Peter Lauritzen et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 251–311. ISBN: 978-3-642-11640-7. DOI: 10.1007/978-3-642-11640-7_9.

[91] Cuong Nguyen, Sebastien Terrana, and Jaime Peraire. “Wall-resolved implicit large eddy simulation of transonic buffet over the OAT15A airfoil using a discontinuous Galerkin method”. In: *AIAA Scitech 2020 Forum*. 2020. DOI: 10.2514/6.2020-2062.

[92] A. Nigro et al. “Modified extended BDF scheme for the discontinuous Galerkin solution of unsteady compressible flows”. In: *International Journal for Numerical Methods in Fluids* 76.9 (2014), pp. 549–574. DOI: https://doi.org/10.1002/fld.3944.

[93] Yvan Notay. “Aggregation-based algebraic multigrid for convection-diffusion equations”. In: *SIAM journal on scientific computing* 34.4 (2012), A2288–A2316. DOI: 10.1137/110835347.

[94] Steven A Orszag. “Spectral methods for problems in complex geometries”. In: *Journal of Computational Physics* 37.1 (1980), pp. 70–92. DOI: 10.1016/0021-9991(80)90005-4.

[95] Steven A. Orszag, Moshe Israeli, and Michel O. Deville. "Boundary conditions for incompressible flows". In: *Journal of Scientific Computing* 1.1 (1986), pp. 75–111. DOI: 10.1007/bf01061454.

[96] Anthony T Patera. "A spectral element method for fluid dynamics: Laminar flow in a channel expansion". In: *Journal of Computational Physics* 54.3 (1984), pp. 468–488. ISSN: 0021-9991. DOI: 10.1016/0021-9991(84)90128-1.

[97] Luca F Pavarino. "Additive Schwarz methods for the p-version finite element method". In: *Numerische Mathematik* 66.1 (1993), pp. 493–515. DOI: 10.1007/bf01385709.

[98] Will Pazner. "Efficient low-order refined preconditioners for high-order matrix-free continuous and discontinuous Galerkin methods". In: *SIAM Journal on Scientific Computing* 42.5 (2020), A3055–A3083. DOI: 10.1137/19M1282052.

[99] Will Pazner, Michael Franco, and Per-Olof Persson. "High-order wall-resolved large eddy simulation of transonic buffet on the OAT15A airfoil". In: *AIAA Scitech 2019 Forum*. 2019. DOI: 10.2514/6.2019-1152.

[100] Will Pazner and Per-Olof Persson. "Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods". In: *Journal of Computational Physics* 354 (2018), pp. 344–369. DOI: 10.1016/j.jcp.2017.10.030.

[101] Will Pazner and Per-Olof Persson. "Interior penalty tensor-product preconditioners for high-order discontinuous Galerkin discretizations". In: *2018 AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, Jan. 2018. DOI: 10.2514/6.2018-1093.

[102] Will Pazner and Per-Olof Persson. "Stage-parallel fully implicit Runge–Kutta solvers for discontinuous Galerkin fluid simulations". In: *Journal of Computational Physics* 335 (2017), pp. 700–717. DOI: 10.1016/j.jcp.2017.01.050.

[103] Jaime Peraire and Per-Olof Persson. "The compact discontinuous Galerkin (CDG) method for elliptic problems". In: *SIAM Journal on Scientific Computing* 30.4 (2008), pp. 1806–1824. DOI: 10.1137/070685518.

[104] Per-Olof Persson and Jaime Peraire. "An efficient low memory implicit DG algorithm for time dependent problems". In: *44th AIAA Aerospace Sciences Meeting and Exhibit*. 2006, p. 113. DOI: 10.2514/6.2006-113.

[105] Per-Olof Persson and Jaime Peraire. "Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier–Stokes equations". In: *SIAM Journal on Scientific Computing* 30.6 (2008), pp. 2709–2733. DOI: 10.1137/070692108.

[106] Andreas Prohl. *Projection and quasi-compressibility methods for solving the incompressible Navier-Stokes equations*. Vieweg+Teubner Verlag, 1997. DOI: 10.1007/978-3-663-11171-9.

[107] Joachim Rang. “Design of DIRK schemes for solving the Navier-Stokes equations”. In: *Informatik-Berichte der Technischen Universität Braunschweig* 2007-02 (2007). URL: http://www.digibib.tu-bs.de/?docid=00020655.

[108] William H Reed and Thomas R Hill. *Triangular mesh methods for the neutron transport equation*. Tech. rep. Los Alamos Scientific Lab., N. Mex.(USA), 1973. URL: https://www.osti.gov/biblio/4491151.

[109] Wim M. van Rees et al. “A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers”. In: *Journal of Computational Physics* 230.8 (Apr. 2011), pp. 2794–2805. DOI: 10.1016/j.jcp.2010.11.031.

[110] Philip L Roe. “Approximate Riemann solvers, parameter vectors, and difference schemes”. In: *Journal of computational physics* 43.2 (1981), pp. 357–372. DOI: 10.1016/0021-9991(81)90128-5.

[111] Philip L Roe. “The use of the Riemann problem in finite difference schemes”. In: *Seventh International Conference on Numerical Methods in Fluid Dynamics*. Springer. 1989, pp. 354–359. DOI: 10.1007/3-540-10694-4_54.

[112] Johann Rudi et al. “An extreme-scale implicit solver for complex PDEs: Highly heterogeneous flow in Earth’s mantle”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’15. Austin, Texas: ACM, 2015, 5:1–5:12. ISBN: 978-1-4503-3723-6. DOI: 10.1145/2807591.2807675.

[113] Christopher L Rumsey et al. “Summary of the 2004 computational fluid dynamics validation workshop on synthetic jets”. In: *AIAA Journal* 44.2 (2006), pp. 194–207. DOI: 10.2514/1.12957.

[114] Youcef Saad. “A flexible inner-outer preconditioned GMRES algorithm”. In: *SIAM Journal on Scientific Computing* 14.2 (Mar. 1993), pp. 461–469. DOI: 10.1137/0914028.

[115] Youcef Saad and Martin H Schultz. “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: *SIAM Journal on scientific and statistical computing* 7.3 (1986), pp. 856–869. DOI: 10.1137/0907058.

[116] Yousef Saad. *Iterative methods for sparse linear systems*. Vol. 82. SIAM, 2003. DOI: 10.1137/1.9780898718003.

[117] Yousef Saad and Jun Zhang. “BILUM: Block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems”. In: *SIAM Journal on Scientific Computing* 20.6 (1999), pp. 2103–2121. DOI: 10.1137/S106482759732753X.

[118] Makoto Sato et al. “Massive parametric study by LES on separated-flow control around airfoil using DBD plasma actuator at Reynolds number 63,000”. In: *43rd AIAA fluid dynamics conference*. 2013. DOI: 10.2514/6.2013-2750.

[119] Jie Shen, Tao Tang, and Li-Lian Wang. *Spectral methods*. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-540-71041-7.

[120] Jonathan Richard Shewchuk. *An introduction to the conjugate gradient method without the agonizing pain*. Tech. rep. 1994. URL: https://dl.acm.org/doi/10.5555/865018.

[121] Chi-Wang Shu. "TVB uniformly high-order schemes for conservation laws". In: *Mathematics of Computation* 49.179 (1987), pp. 105–121. DOI: 10.1090/S0025-5718-1987-0890256-5.

[122] Chi-Wang Shu et al. "Numerical convergence study of nearly incompressible, inviscid Taylor-Green vortex flow". In: *Journal of Scientific Computing* 24.1 (July 2005), pp. 1–27. DOI: 10.1007/s10915-004-5407-y.

[123] Philippe R Spalart. "Comments on the feasibility of LES for wings, and on a hybrid RANS/LES approach". In: *Proceedings of first AFOSR international conference on DNS/LES*. Greyden Press. 1997. URL: https://www.researchgate.net/publication/236888805_Comments_on_the_Feasibility_of_LES_for_Wings_and_on_a_Hybrid_RANSLES_Approach.

[124] Alexey Stomakhin et al. "A material point method for snow simulation". In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–10. DOI: 10.1145/2461912.2461948.

[125] Kasia Świrydowicz et al. "Acceleration of tensor-product operations for high-order finite element methods". In: *The International Journal of High Performance Computing Applications* 33.4 (2019), pp. 735–757. DOI: 10.1177/1094342018816368.

[126] Saul A. Teukolsky. "Short note on the mass matrix for Gauss-Lobatto grid points". In: *Journal of Computational Physics* 283 (Feb. 2015), pp. 408–413. DOI: 10.1016/j.jcp.2014.12.012.

[127] A. G. Tomboulides, J. C. Y. Lee, and S. A. Orszag. "Numerical simulation of low Mach number reactive flows". In: *Journal of Scientific Computing* 12.2 (June 1997), pp. 139–167. ISSN: 1573-7691. DOI: 10.1023/A:1025669715376.

[128] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*. Vol. 34. Springer Science & Business Media, 2005. DOI: 10.1007/b137868.

[129] PG Tucker. "Computation of unsteady turbomachinery flows: Part 1—Progress and challenges". In: *Progress in Aerospace Sciences* 47.7 (2011), pp. 522–545. DOI: 10.1016/j.paerosci.2011.06.004.

[130] A Uranga et al. "Implicit large eddy simulation of transition to turbulence at low Reynolds numbers using a discontinuous Galerkin method". In: *International Journal for Numerical Methods in Engineering* 87.1-5 (2011), pp. 232–261. DOI: 10.1002/nme.3036.

[131] Brian C Vermeire, Freddie D Witherden, and Peter E Vincent. "On the utility of GPU accelerated high-order methods for unsteady flow simulations: A comparison with industry-standard tools". In: *Journal of Computational Physics* 334 (2017), pp. 497–521. DOI: 10.1016/j.jcp.2016.12.049.

[132] Zhijian J Wang et al. "High-order CFD methods: current status and perspective". In: *International Journal for Numerical Methods in Fluids* 72.8 (2013), pp. 811–845. DOI: 10.1002/fld.3767.

[133] ZJ Wang and Salman Rahmani. "Implicit large eddy simulation of the NASA CRM high-lift configuration near stall". In: *Computers & Fluids* 220 (2021), p. 104887. DOI: 10.1016/j.compfluid.2021.104887.

[134] TC Warburton and George Em Karniadakis. "A discontinuous Galerkin method for the viscous MHD equations". In: *Journal of computational Physics* 152.2 (1999), pp. 608–641. DOI: 10.1006/jcph.1999.6248.

[135] Dirk Wilhelm and Leonhard Kleiser. "Domain decomposition method and fast diagonalization solver for spectral element simulations". In: *Computational Fluid Dynamics 2000* (2001), pp. 429–434. DOI: 10.1007/978-3-642-56535-9_64.

[136] Jinchao Xu. "Iterative methods by space decomposition and subspace correction". In: *SIAM review* 34.4 (1992), pp. 581–613. DOI: 10.1137/1034116.

[137] Ulrike Meier Yang et al. "BoomerAMG: A parallel algebraic multigrid solver and preconditioner". In: *Applied Numerical Mathematics* 41.1 (2002), pp. 155–177. DOI: 10.1016/S0168-9274(01)00115-5.

Appendix A

Time integration schemes

We present the specific time integration schemes used throughout this thesis.

A.1 Diagonally implicit Runge-Kutta schemes

We use the two-stage, second-order DIRK22 scheme and three-stage, third-order DIRK33 schemes derived by [1]. These schemes are A-stable and S-stable, and thus well suited to stiff system of ODE. They are also singly DIRK schemes, in that all diagonal coefficients a_{ii} are identical, allowing for the efficient re-use of preconditioners. The DIRK22 Butcher tableau is presented in Table A.1. The DIRK33 Butcher tableau is presented in Table A.2.

$1+\frac{\sqrt{2}}{2}$	$1+\frac{\sqrt{2}}{2}$	0
1	$-\frac{\sqrt{2}}{2}$	$1+\frac{\sqrt{2}}{2}$
	$-\frac{\sqrt{2}}{2}$	$1+\frac{\sqrt{2}}{2}$

Table A.1: DIRK22 scheme

A.2 Backward differentiation formulae

We use the two-step BDF method BDF2 [68], defined as

$$\boldsymbol{u}_{n+2} - \frac{4}{3}\boldsymbol{u}_{n+1} + \frac{1}{3}\boldsymbol{u}_n = \frac{2}{3}\Delta t f\left(t_{n+2}, \boldsymbol{u}_{n+2}\right). \tag{A.1}$$

BDF2 is A-stable and second-order accurate.

$$
\begin{array}{c|ccc}
\alpha & \alpha & 0 & 0 \\
\frac{1+\alpha}{2} & \frac{1-\alpha}{2} & \alpha & 0 \\
1 & b_1 & b_2 & \alpha \\
\hline
 & b_1 & b_2 & \alpha
\end{array}
$$

Table A.2: DIRK33 scheme

$$
\begin{aligned}
\alpha &= 1 + \frac{\sqrt{6}}{2} \sin\left(\frac{1}{3}\arctan\left(\frac{\sqrt{2}}{4}\right)\right) - \frac{\sqrt{2}}{2} \cos\left(\frac{1}{3}\arctan\left(\frac{\sqrt{2}}{4}\right)\right) \\
b_1 &= -\frac{1}{4}\left(6\alpha^2 - 16\alpha + 1\right) \\
b_2 &= \frac{1}{4}\left(6\alpha^2 - 20\alpha + 5\right)
\end{aligned}
$$

We also use the three-step BDF method BDF3 [68], defined as

$$
\boldsymbol{u}_{n+3} - \frac{18}{11}\boldsymbol{u}_{n+2} + \frac{9}{11}\boldsymbol{u}_{n+1} - \frac{2}{11}\boldsymbol{u}_n = \frac{6}{11}\Delta t f\left(t_{n+3}, \boldsymbol{u}_{n+3}\right). \tag{A.2}
$$

BDF3 is third-order accurate, A(α)-stable for $\alpha = 86.03°$.