WARSAW UNIVERSITY OF TECHNOLOGY

Ph.D. Thesis

Mechanical Engineering/
Engineering and Technology

Jakub Gałecki M.Sc.

Efficient and Scalable Application of the Least-Squares Spectral/*hp* Element Method to Incompressible Flow Problems

Supervisor
prof. **Jacek Szumbarski**

WARSAW 2025

Streszczenie

Metoda elementów skończonych w ujęciu najmniejszych kwadratów (MESNK) jest sposobem numerycznego rozwiązywania równań różniczkowych cząstkowych, oferującym szereg zalet nad klasycznym sformułowaniem Galerkina. Praca opisuje aplikację MESNK do zagadnień przepływów nieściśliwych, kładąc nacisk na aspekty obliczeniowe tego podejścia. Schemat rozprzęgający składowe rozwiązania został omówiony i rozszerzony o otwarte wylotowe warunki brzegowe. Zaprezentowano różne strategie ewaluacji algebraicznych operatorów wynikających z zastosowania dyskretyzacji wysokiego rzędu do sformułowania najmniejszych kwadratów, w tym wyprowadzono technikę faktoryzacji sumy dla operatorów MESNK. Strategie te poddano systematycznej analizie, przeprowadzono także studium obliczeniowe ich wydajności. Metody i algorytmy prezentowane w pracy zaimplementowano w ramach otwartego kodu obliczeniowego L3STER. Pokazano przykłady zastosowania solwera, w tym studium zbieżności w czasie, bezpośrednią numeryczną symulację przepływu wokół cylindra dla liczby Reynoldsa $Re = 400$, a także symulacje przepływu w szczelinie skalnej. Wykazano, że zaproponowana technika faktoryzacji oferuje wyższą niż alternatywne podejścia wydajność dla MESNK, szczególnie dla wysokich rzędów aproksymacji i zagadnień trójwymiarowych.

Słowa kluczowe: metoda elementów skończonych w ujęciu najmniejszych kwadratów, metoda elementów spektralnych/*hp*, przepływy nieściśliwe, bezmacierzowa ewaluacja operatora, faktoryzacja sumy, obliczenia dużej mocy.

Abstract

The least-squares finite element method (LSFEM) is a numerical method for the solution of partial differential equations, which offers several advantages compared to the classic Galerkin formulation. In this work, the LSFEM is applied to incompressible flow problems, with a focus on the computational efficiency of the approach. A pressure-correction-type splitting scheme for the solution components is reviewed and extended to include open boundary conditions. Strategies for the evaluation of the algebraic operators arising from the application of a high-order discretization to the least-squares formulation are discussed, including the derivation of the sum-factorization technique for LSFEM operators. The different approaches are systematically analyzed, and a computational study of their performance is carried out. The methods and algorithms discussed are implemented within an open-source numerical code. Examples of applications of the solver are presented, including time convergence studies, direct numerical simulation of flow past a cylinder at the Reynolds number $Re = 400$, and flow in a fracture. It is shown that the sum-factorization technique offers superior performance within the context of the LSFEM, particularly at higher approximation orders and in three dimensions.

Keywords: least-squares finite element method, spectral/*hp* element method, incompressible flow, matrix-free operator evaluation, sum-factorization, high-performance computing.

Acknowledgments

The author wishes to thank the following colleagues: Łukasz Łaniewski-Wołłk, for sharing his valuable insights and opening the door to interesting research opportunities, Stanisław Gepner for his encouragement and introduction to fellow SEM practitioners, Daria Żyła-Jabłońska for her feedback on the developed code, and Michał Dzikowski for his various tips and suggestions. Many thanks go to professor Spencer Sherwin, Dave Moxey, and Martin Kronbichler for their technical advice and inspiration given at conferences, which in part shaped the topic of the present work. Last, but certainly not least, the author wishes to express his gratitude to professor Jacek Szumbarski for his sustained mentorship over the years.

Simulations performed in the course of this work were in part conducted at the Interdisciplinary Center for Mathematical and Computational Modeling of the Warsaw University (ICM UW), under computational grant number g101-2387. The author thanks ICM UW for the provided resources.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Computational Fluid Dynamics (CFD) has emerged as one of the key tools of modern science and engineering, enabling the study and prediction of fluid flows across a wide range of applications, including aerospace and automotive design, biomedical engineering, climate modeling, energy systems, and geophysical sciences. By solving the governing equations of fluid motion numerically, CFD provides an alternative to costly and time-consuming experiments, while additionally offering insights into complex flow phenomena that are inaccessible to direct measurement. The roots of CFD can be traced back to the 20th century, when advances in computing first made it possible to approximate solutions of the partial differential equations governing fluid dynamics. Early efforts, such as those by John von Neumann and his contemporaries in the 1940s and 1950s, demonstrated the potential of numerical methods for solving simplified fluid flow problems [47]. These foundational studies laid the groundwork for the development of more sophisticated numerical models in the 1960s, using aproaches such as the panel method, and finite difference, volume, and element methods. By the 1970s and 1980s, as computing power rapidly grew, CFD matured into a discipline in its own right. Today, CFD resides at the intersection of fluid mechanics, applied mathematics, and computer science. Conducting research in this discipline often requires a combination of knowledge of the underlying physics and numerical methods, but also programming skills and high-performance computing experience. Modern codes used for CFD simulations must exhibit high accuracy, stability, and performance, as well as scalability and the ability to efficiently run on increasingly

heterogeneous hardware.

High-order methods, such as the spectral and spectral/*hp* element methods, have emerged as attractive tools for discretization, offering very high accuracy [20]. Within this discretization framework, different mathematical formulations can be used, such as the continuous and discontinuous Galerkin and Petrov-Galerkin formulations. An interesting member of this class, and of particular interest here, is the least-squares formulation, which exhibits several desirable features, such as the symmetric positive-definite nature of the resulting algebraic systems and circumvention of compatibility conditions imposed on the approximation spaces used for the pressure and velocity. The application of this approach to incompressible flow problems constitutes the topic of the present work.

1.2 State of the Art

Single-domain spectral methods have a rich history as a tool for CFD simulations, their foundational ideas dating back to the 1940s. Their major development occurred in the 1970s and 80s, with the works of Gottlieb and Orszag [73, 36] and Canuto et al. [14]. The spectral element method (SEM), which extends the spectral method to a multi-domain setting, was introduced by Patera [79], who later, together with Rønquist, developed the nodal-basis approach [92]. The spectral/*hp* element method combines these ideas with the works on high-order finite element methods (FEM) by Szabó and Babuška [98]. This approach benefits both from the accuracy of spectral methods and the geometric flexibility of the FEM. It has since greatly gained in popularity, with new mathematical formulations, such as discontinuous Galerkin [16] and discontinuous Petrov-Galerkin [19] being proposed. The spectral/*hp* element method has found further success with the advent of modern computing. One of the features of high-order methods is their high arithmetic density, which allows for efficient employment on modern architectures, where the memory bandwidth is significantly smaller than the peak computing power [60, 61]. The key observation enabling this approach was first made by Orszag [74] and revolves around the tensor-product structure of the basis functions and quadrature abscissas. The sum-factorization technique has since been developed to fully take advantage of this structure, and numerous computational studies undertaken to further refine its implementation for various formulations, element types, and architectures [11, 13, 28, 59, 70, 107]. Several

mature software frameworks for the spectral/*hp* element method exist, such as MFEM [1], Deal.II [3], and Nektar++ [12].

The least-squares variational formulation and associated least-squares finite element method (LSFEM) was developed in an effort to recover the Rayleigh-Ritz setting for equations with non-self-adjoint operators. It is distinct from the classic (Bubnov-)Galerkin method, and relies on the minimization of an error norm, not the posing of the problem in a weak form. The LSFEM traces its roots to the works of Wendland [108] and Aziz [4]. The first monograph on the subject, containing practical recommendations and applications, was published by Jiang in 1998 [52]. A comprehensive mathematical theory underpinning the LSFEM was subsequently established by Bochev and Gunzburger, culminating in their seminal work from 2009 [40]. The LSFEM has been developed for incompressible flow problems in the works of Pontaza, Prabhakar, Reddy, and Vallalaa [81, 82, 84, 85, 86, 87, 88, 106]. The cited works present new formulations of the Navier-Stokes equations and address some of the shortcomings of the LSFEM, such as poor mass conservation. Notable contributions have also been made by Gerristma et al. [29, 69, 76, 77, 89]. A key paper demonstrating a consistent splitting scheme enabling the efficient solution of transient problems was published by Pontaza in 2007 [83]. Further contributions have been made by Heys et al. [45, 44]. The application of multigrid solvers to the resulting algebraic systems has been studied in [72, 75, 90]. More recently, the LSFEM has been employed in the works [5, 31, 62].

1.3 Aims and Scope

Despite a large part of the LSFEM canon, including the majority of the works cited above, operating in the high-order setting, issues surrounding computational efficiency of the various proposed approaches are seldom discussed. To the extent of the author's knowledge, the sum-factorization technique, essential for the performance of other types of SEM, has never been considered in the least-squares context. This is a key research gap which this work aims to bridge. The research hypothesis of this thesis is that the least-squares spectral/*hp* element method can be applied to solve incompressible flow problems efficiently. Consequently, the aim of the present work is to develop efficient and scalable algorithms for the solution of incompressible flow problems using the LSFEM.

Furthermore, the aforementioned splitting scheme from [83] will be investigated in three dimensions and for boundary conditions not discussed in the original paper. The efficiency of the proposed approaches will be analyzed and verified computationally, and examples of applications shown. Their implementation is made available as part of an open-source framework named L3STER, which should be considered an integral element of the present work, and can serve other researchers in the future.

The text is organized as follows. In chapter 2, the governing equations of incompressible flow are reviewed and converted into a form suitable for the application of the LSFEM. Chapter 3 discusses the least-squares formulation and the high-order discretization employed. Issues surrounding the coordinate transformation from reference elements and numerical integration are also addressed. Chapter 4 focuses on the different operator evaluation strategies which can be used in the course of iteratively solving the algebraic system, and which determine the performance of the solver. Their computational cost is investigated theoretically. In chapter 5, this matter is further studied computationally, allowing for more practical recommendations. The details of the implementation are discussed, as well as some considerations relating to software design. Examples of applications of the proposed approach and further computational studies are presented in chapter 6. Chapter 7 contains the concluding remarks.

Chapter 2

Governing Equations

This chapter describes the foundational equations governing and associated with incompressible fluid flow. These equations can be found in most fluid dynamics textbooks, such as [91]. In order to conform to the constraints imposed by the least-squares finite element method, described in the next chapter, the equations are converted to their equivalent first-order forms. The issues of linearization and time-discretization are also addressed. All equations contained in this chapter are partial differential equations (PDEs) defined in the domain Ω. The domain is assumed to be an open bounded subset of $\mathbb{R}^{N_d}$, where $N_d \in \{2, 3\}$ is the spatial dimension of the problem. The boundary of Ω is denoted by $\partial\Omega = \bar{\Omega} \setminus \Omega$ and is assumed to be Lipschitz continuous. Unsteady equations are additionally defined in the time interval $(0, T]$, T being the time bound.

2.1 Advection-Diffusion Equation

The advection-diffusion equation describes the transport of the passive scalar quantity $\phi : \Omega \times (0, T] \to \mathbb{R}$. The transport takes place in the flow field defined by the advection velocity $\mathbf{a} : \Omega \times (0, T] \to \mathbb{R}^{N_d}$. ϕ diffuses according to the diffusivity coefficient $\lambda : \Omega \times (0, T] \to \mathbb{R}$. Additionally, ϕ may be created or lost according to the local source/sink term $s : \Omega \times (0, T] \to \mathbb{R}$. Formally, the above can be expressed in the following form:

$$\frac{\partial \phi}{\partial t} + (\mathbf{a} \cdot \nabla)\, \phi - \nabla \cdot (\lambda \nabla \phi) = s \quad \text{in } \Omega \times (0, T]\,. \tag{2.1}$$

Equation (2.1) must be closed with appropriate boundary conditions (BCs). In the present work, the following Dirichlet and Neumann BCs are considered:

$$\phi = \phi_D \qquad \text{on } \Gamma_D, \tag{2.2a}$$

$$\frac{\partial \phi}{\partial \mathbf{n}} = s_N \quad \text{on } \Gamma_N, \tag{2.2b}$$

where $\partial\Omega = \Gamma_D \cup \Gamma_N$, $\phi_D : \Gamma_D \times (0,T] \to \mathbb{R}$ is the value of ϕ prescribed at Γ_D, and $s_N : \Gamma_N \times (0,T] \to \mathbb{R}$ is the flux of ϕ prescribed at Γ_N. Here, $\frac{\partial \phi}{\partial \mathbf{n}} = \mathbf{n} \cdot \nabla \phi$ denotes the normal derivative, $\mathbf{n}$ being the outward-facing boundary unit normal. Finally, the initial condition is given as

$$\phi(\mathbf{x}, 0) = \phi_0, \tag{2.3}$$

for some known $\phi_0 : \Omega \to \mathbb{R}$. Note that if λ is constant, the diffusive term in (2.1) can be simplified to $\nabla \cdot (\lambda \nabla \phi) = \lambda \Delta \phi$.

In order to recast the equation (2.1) into its equivalent first-order form, the gradient of λ, denoted by $\mathbf{q} : \Omega \times (0,T] \to \mathbb{R}^{N_d}$, is introduced as an auxiliary unknown:

$$\begin{cases} \dfrac{\partial \phi}{\partial t} + (\mathbf{a} \cdot \nabla)\, \phi - \nabla \lambda \cdot \mathbf{q} - \lambda \nabla \cdot \mathbf{q} = s & \text{(2.4a)} \\ \mathbf{q} - \nabla \phi = \mathbf{0} & \text{(2.4b)} \\ \nabla \times \mathbf{q} = \mathbf{0} & \text{(2.4c)} \end{cases} \qquad \text{in } \Omega \times (0,T].$$

Note that the seemingly redundant equation (2.4c) is present in order to make the resulting operator coercive[1] [40, 52]. The corresponding boundary conditions are given as

$$\phi = \phi_D \qquad \text{on } \Gamma_D, \tag{2.5a}$$

$$\mathbf{n} \cdot \mathbf{q} = s_N \quad \text{on } \Gamma_N. \tag{2.5b}$$

The initial condition (2.3) remains unchanged.

2.2 Incompressible Navier-Stokes Equations

The incompressible Navier-Stokes equations are a set of key PDEs in the field of continuum mechanics. The proof of the existence and smoothness of an analytical solution remains

[1] Coercivity is a property of the operator which expresses that its action is bounded from below. Its presence guarantees desirable properties of the numerical method used in the present work. A deeper exploration of the functional analysis framework underpinning the method exceeds the scope of the present text. An interested reader may review the sections pertaining to Agmon–Douglis–Nirenberg theory in the references cited above.

one of the great open problems of mathematics, and has been included in the set of Millenium Problems established by the Clay Mathematics Institute [26]. The equations describe the motion of a viscous, incompressible fluid. Physically, they represent the conservation of momentum and mass. The momentum equation can be expressed as

$$\rho \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{f}, \tag{2.6}$$

where $\rho \in \mathbb{R}$ is the density, $\mathbf{u} : \Omega \times [0, T) \to \mathbb{R}^{N_d}$ is the velocity, $\boldsymbol{\sigma} : \Omega \times [0, T) \to \mathbb{R}^{N_d \times N_d}$ is the stress tensor, and $\mathbf{f} : \Omega \times [0, T) \to \mathbb{R}^{N_d}$ is the known external body force. Here, the notation $\frac{\mathrm{D}}{\mathrm{D}t}$ represents the material derivative. For an incompressible, Newtonian fluid the stress tensor is given by

$$\boldsymbol{\sigma} = -p\mathbf{I} + \mu \left(\nabla \mathbf{u} + \nabla^T \mathbf{u} \right), \tag{2.7}$$

where $\mu \in \mathbb{R}$ is the dynamic viscosity, $p : \Omega \times [0, T) \to \mathbb{R}$ is the pressure, and $\mathbf{I}$ denotes the identity matrix. Mass conservation is achieved by requiring that the time derivative of the density be equal to the divergence of $\rho\mathbf{u}$. Since density is assumed constant, this simplifies to requiring that the velocity field be solenoidal. Combining this condition together with (2.6) and (2.7), the incompressible Navier-Stokes equations can be posed as

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\, \mathbf{u} - \nu \Delta \mathbf{u} + \nabla p = \mathbf{f} & \text{in } \Omega, \quad (2.8a) \\ \nabla \cdot \mathbf{u} = 0 & (2.8b) \end{cases}$$

where $\nu = \frac{\mu}{\rho}$ denotes the kinematic viscosity of the fluid. Note that in (2.8), and throughout the remainder of this work, the pressure has been normalized by the constant density, eliminating the latter from the equations. The unit of p is therefore $\frac{m^2}{s^2}$, not Pa.

The system (2.8) must be closed with appropriate boundary conditions. In the present work, the following BCs are considered:

$$\mathbf{u} = \mathbf{u}_D \qquad \text{on } \Gamma_D, \tag{2.9a}$$

$$-p\mathbf{n} + \nu \mathbf{n} \cdot \nabla \mathbf{u} = \mathbf{g} \quad \text{on } \Gamma_O, \tag{2.9b}$$

where $\mathbf{u}_D : \Gamma_D \to \mathbb{R}^{N_d}$ and $\mathbf{g} : \Gamma_O \to \mathbb{R}^{N_d}$ are given, and Γ_D and Γ_O are distinct parts of the domain boundary, such that $\partial\Omega = \Gamma_D \cup \Gamma_O$. The initial condition for the velocity reads

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0, \tag{2.10}$$

for some known $\mathbf{u}_0 : \Omega \to \mathbb{R}^{N_d}$.

Let us now comment on the boundary conditions (2.9). The Dirichlet condition (2.9a) can be imposed on parts of the boundary where the velocity is known, such as no-slip walls or, possibly, inlets. The BC (2.9b), most often referred to as the open boundary condition[2], is a type of outflow boundary condition. Such BCs are needed when a (possibly very large, or even unbounded) physical region is truncated in order to create the computational domain. For example, when simulating flow past a bluff body, typically only the immediate vicinity of the obstacle is of interest, e.g., in order to compute the forces acting on the submerged object. Extending the domain a long distance past the body is therefore computationally wasteful. Properly formulated outflow boundary conditions allow the fluid to discharge from the domain without affecting the upstream flow in non-physical ways. This area has been the subject of research for several decades, some notable publications on this subject include [8, 68, 95, 104]. More recently, boundary conditions which address the issue of backflow instability have been proposed [21, 22, 23]. Backflow instability is a phenomenon wherein a local reversal of the flow at the outflow boundary may precipitate an uncontrolled growth in the local kinetic energy, ultimately causing the entire simulation to fail. Employing such stabilized BCs allows for a more aggressive truncation of the domain, and hence computational savings, since vortices do not need to travel as far before dissipating sufficiently for backflow not to occur when they exit the domain. It is worth noting that open boundary conditions set the reference pressure at the outlet on which they are defined. When multiple outlet sections are present, specifying different pressure levels at different outlets via the source term $\mathbf{g}$ on the right-hand side of (2.9b) (or analogous open BC) can be used to drive the flow [31, 100]. In the present work, $\mathbf{g}$ is only used for the purposes of numerical testing, and is set to $\mathbf{g} \equiv \mathbf{0}$ otherwise. Furthermore, if $\Gamma_O = \emptyset$, the pressure level must be set by imposing a Dirichlet condition on the pressure at some point in the domain, otherwise p is defined only up to a constant.

The Reynolds number Re is a key dimensionless parameter used in the description of fluid flow. It expresses the relative importance of inertial and viscous forces in the flow dynamics. If we denote by U the reference velocity, and by L the reference length scale

[2]The BC (2.9b) is sometimes also called the pseudo-traction condition. This is due to the formal similarity to the traction condition, which operates on the normal stress $\mathbf{n} \cdot \boldsymbol{\sigma}$. Indeed, omitting the term involving $\nabla^T \mathbf{u}$ from the stress tensor leads directly to (2.9b).

(e.g. the width of a channel or height of an obstacle), the Reynolds number is equal to

$$Re = \frac{LU}{\nu}. \tag{2.11}$$

Low values of Re indicate the dominance of viscous forces, while at high Re inertial effects play a larger role. With the help of this parameter, the Navier-Stokes equations can also be expressed in their non-dimensional form. Indeed, by normalizing the quantities present in (2.8) using U and L, one can derive the dimensionless system, wherein ν is replaced by $\frac{1}{Re}$. The Stokes equations are an approximation of the Navier-Stokes system, commonly used at low Re. These are obtained by omitting the inertial term $(\mathbf{u} \cdot \nabla)\,\mathbf{u}$, effectively substituting the material derivative with the partial time derivative.

In order to convert the Navier-Stokes equations to their equivalent first-order form, auxiliary variables representing the derivatives of the velocity must be introduced. However, unlike the advection-diffusion equation, not all these derivatives need be present. Popular formulations are based on other physical quantities, such as the stress tensor [106]. In the present work, the commonly used velocity-vorticity-pressure ($u - \omega - p$) formulation is employed [52, 85]. It can be derived with the help of the well-known vector identity

$$\Delta \mathbf{u} = \nabla\,(\nabla \cdot \mathbf{u}) - \nabla \times (\nabla \times \mathbf{u})\,. \tag{2.12}$$

Since $\mathbf{u}$ is solenoidal, the first term on the right hand side of (2.12) must be zero. Introducing the vorticity $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ as an additional unknown yields

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} + \nu \nabla \times \boldsymbol{\omega} + \nabla p = \mathbf{f} & \text{(2.13a)} \\ \nabla \cdot \mathbf{u} = 0 & \text{(2.13b)} \\ \boldsymbol{\omega} - \nabla \times \mathbf{u} = \mathbf{0} & \text{(2.13c)} \\ \nabla \cdot \boldsymbol{\omega} = 0 & \text{(2.13d)} \end{cases} \quad \text{in } \Omega.$$

The equation (2.13d) is present to ensure the coerciveness of the associated operator, similarly to (2.4c) in the case of the first-order advection-diffusion system. It is tautologically satisfied if $N_d = 2$. The boundary and initial conditions remain unaltered. It is worth emphasizing that the $u - \omega - p$ formulation introduces only 1 and 3 additional unknowns in two and three dimensions, respectively, compared to N_d^2 auxiliary unknowns if the entire $\nabla \mathbf{u}$ tensor were naively used instead.

The Navier-Stokes equations contain the nonlinear convective term $(\mathbf{u} \cdot \nabla)\,\mathbf{u}$. However, methods widely used to numerically solve PDEs, including the one which is the

subject of the present work, require the equations to be linear. Additional treatment of the system (2.8) is therefore required. Two commonly used approaches to linearization are the Picard and Newton methods. Both of them require some guess for the velocity, denoted by $\mathbf{u}_\star$, ideally very close to the unknown velocity $\mathbf{u}$. This guess can be obtained, e.g., by extrapolation in time. If the problem being solved is steady and no good approximation is otherwise available, $\mathbf{u}_\star \equiv \mathbf{0}$ can be used, which is equivalent to solving the Stokes problem. The Picard method consists of simply setting the advection velocity to $\mathbf{u}_\star$, giving rise to the following approximation:

$$(\mathbf{u} \cdot \nabla)\,\mathbf{u} \approx (\mathbf{u}_\star \cdot \nabla)\,\mathbf{u}. \tag{2.14}$$

In the Newton method, the approximation is obtained by a first-order Taylor series expansion about $\mathbf{u}_\star$, resulting in

$$(\mathbf{u} \cdot \nabla)\,\mathbf{u} \approx (\mathbf{u}_\star \cdot \nabla)\,\mathbf{u} + (\mathbf{u} \cdot \nabla)\,\mathbf{u}_\star - (\mathbf{u}_\star \cdot \nabla)\,\mathbf{u}_\star. \tag{2.15}$$

The Newton method boasts better accuracy, however it possesses a smaller convergence radius than the Picard method. Unless stated otherwise, the former is used throughout the present work, as sufficiently accurate guesses for the velocity are available in all application examples presented in chapter 6 which involve nonlinearity. If the results obtained from solving the linear equation are unsatisfactory, multiple iterations can be used, where $\mathbf{u}_\star$ is set to the velocity obtained in the previous iteration.

An alternative strategy for handling nonlinearity in time-dependent problems is the Operator-Integration-Factor Splitting technique (OIFS). While this approach is not explored in the present work, an interested reader may find a study of its application to incompressible flow in [66]. More recent developments are reviewed in [78].

2.3 Time Discretization

All PDEs discussed above describe the evolution of physical quantities in time, and hence involve time derivatives. It is most often impractical to solve space-time coupled systems. For this reason, the problem must be discretized in time, and some form of time-marching procedure must be used. In the present work, we employ a time discretization strategy based on the backward differentiation formulae (BDF) [49], a family of implicit time-stepping schemes. In the BDF of order J, the time derivative of some quantity u at the

time step $n+1$ is approximated using the values of u at $J+1$ time steps (including $n+1$):

$$\frac{\partial u^{n+1}}{\partial t} \approx \frac{\mathrm{d}u^{n+1}}{\Delta t} = \frac{1}{\Delta t}\left(\alpha_0 u^{n+1} - \sum_{i=1}^{J} \alpha_i u^{n+1-i}\right). \tag{2.16}$$

Here, Δt denotes the time increment between two time steps (the time step size). The notation $\frac{\mathrm{d}}{\Delta t}$ describes the discrete time derivative, and the superscript $(\cdot)^n$ is used to indicate the index of the time step. The values of the coefficients $\alpha_i \in \mathbb{R}$ for selected orders J have been compiled in Table 2.1. Applying (2.16) to the time derivatives of ϕ in (2.1) and (2.4) or $\mathbf{u}$ in (2.8) and (2.13) yields systems of PDEs which only involve spatial derivatives.

A topic related to time discretization is time extrapolation, wherein an approximation for the value of u at the time step $n+1$ is sought using only the values of u at previous time steps. This issue can arise, e.g., when formulating a guess for the unknowns within the context of the linearization methods described above. In the extrapolation scheme of order J, the values from J previous time iterations are used. A simple and effective way of constructing the extrapolation is by spanning a Lagrange polynomial over these values and evaluating it at $t = (n+1)\cdot\Delta t$, allowing $u_\star^{n+1}$ to be expressed as the following linear combination:

$$u_\star^{n+1} = \sum_{i=1}^{J} \beta_i u^{n+1-i}, \tag{2.17}$$

where the subscript $(\cdot)_\star$ has been used to indicate the extrapolation. The values of the coefficients $\beta_i \in \mathbb{R}$ for selected orders J have been compiled in Table 2.2.

J	α_0	α_1	α_2	α_3	α_4
1	1	1	-	-	-
2	$\frac{3}{2}$	2	$-\frac{1}{2}$	-	-
3	$\frac{11}{6}$	3	$-\frac{3}{2}$	$\frac{1}{3}$	-
4	$\frac{25}{12}$	4	-3	$\frac{4}{3}$	$-\frac{1}{4}$

Table 2.1. Coefficients of the BDF schemes of different orders J.

J	β_1	β_2	β_3	β_4
1	1	-	-	-
2	2	-1	-	-
3	3	-3	1	-
4	4	-6	4	-1

Table 2.2. Extrapolation coefficients for different orders J.

2.4 Splitting Scheme

The Navier-Stokes problem (2.8) is formulated above in an un-split manner, meaning that it couples all of the solution components. This setting naturally implies a higher solution cost than for scalar problems. This observation applies *a fortiori* to the first-order formulations, which introduce even more unknowns. Furthermore, a major difficulty in the numerical solution of the Navier-Stokes equations lies in the coupling of the velocity and pressure via the continuity equation. To address these issues, intensive research was conducted in the second half of the 20th century, seeking to develop ways of splitting the full system into multiple smaller problems, which could be solved for the individual unknowns. One of the major difficulties lay in formulating a strategy with optimal convergence for the pressure and velocity, meaning a scheme formally of order J which would exhibit $\mathcal{O}\left(\Delta t^J\right)$ error estimates for both solution components. A key work resolving this issue was published by Guermond and Shen [39]. Since then, several such schemes have been developed, such as [24, 101]. The issue of incorporating open boundary conditions within the framework of such strategies has also been addressed [37]. The reader may find a comprehensive review of the different types of projection methods in [38].

In the present work, we choose to employ the consistent splitting scheme proposed by Pontaza in [83], further extending it by inclusion of the open boundary condition (2.9b). This choice is motivated by the following factors: (i) the scheme is tailored towards the least-squares finite element method, (ii) it offers excellent mass conservation resulting from the explicit enforcement of the continuity equation, (iii) its author claims an "enhanced" convergence rate. The latter issue in particular is investigated in chapter 6. The structure of the splitting scheme resembles the rotational pressure correction-type strategy with an additional div-curl equation. At the time step $n+1$, given the previous J values of the

velocity and pressure, we compute $\mathbf{u}^{n+1}$ and p^{n+1}, along with an intermediate velocity approximation $\hat{\mathbf{u}}^{n+1}$. Every time iteration consists of the following three phases:

1. For $\hat{\mathbf{u}}^{n+1}$:

$$\frac{\mathrm{d}\hat{\mathbf{u}}^{n+1}}{\Delta t} + \left(\mathbf{u}_\star^{n+1} \cdot \nabla\right) \hat{\mathbf{u}}^{n+1} - \nu \Delta \hat{\mathbf{u}}^{n+1} = -\nabla p_\star^{n+1} + \mathbf{f}^{n+1} \quad \text{in } \Omega, \tag{2.18a}$$

$$\hat{\mathbf{u}}^{n+1} = \mathbf{u}_D^{n+1} \quad \text{on } \Gamma_D, \tag{2.18b}$$

$$\nu \mathbf{n} \cdot \nabla \hat{\mathbf{u}}^{n+1} = p_\star^{n+1} \mathbf{n} + \mathbf{g}^{n+1} \quad \text{on } \Gamma_O. \tag{2.18c}$$

2. For $\mathbf{u}^{n+1}$:

$$\begin{cases} \nabla \times \mathbf{u}^{n+1} = \nabla \times \hat{\mathbf{u}}^{n+1} & \text{in } \Omega \\ \nabla \cdot \mathbf{u}^{n+1} = 0 \end{cases} \tag{2.19a, 2.19b}$$

$$\mathbf{n} \cdot \mathbf{u}^{n+1} = \mathbf{n} \cdot \hat{\mathbf{u}}^{n+1} \quad \text{on } \Gamma_D \cup \Gamma_O \tag{2.19c}$$

3. For p^{n+1}:

$$\Delta p^{n+1} = \nabla \cdot \mathbf{f}^{n+1} - \nabla^T \mathbf{u}^{n+1} : \nabla \mathbf{u}^{n+1} \quad \text{in } \Omega, \tag{2.20a}$$

$$\mathbf{n} \cdot \nabla p^{n+1} = \\ = \mathbf{n} \cdot \left(\mathbf{f}^{n+1} - \frac{\mathrm{d}\mathbf{u}^{n+1}}{\Delta t} - \left(\mathbf{u}^{n+1} \cdot \nabla\right) \mathbf{u}^{n+1} - \nu \nabla \times \left(\nabla \times \mathbf{u}^{n+1}\right) \right) \quad \text{on } \Gamma_D, \tag{2.20b}$$

$$p^{n+1} = \nu \mathbf{n} \cdot \nabla \mathbf{u}^{n+1} \cdot \mathbf{n} - \mathbf{n} \cdot \mathbf{g}^{n+1} - \nu \nabla \cdot \mathbf{u}^{n+1} \quad \text{on } \Gamma_O. \tag{2.20c}$$

The first phase arises from an explicit treatment of the pressure. It involves solving an advection-diffusion system for $\hat{\mathbf{u}}^{n+1}$, where the advection velocity is obtained by extrapolating $\mathbf{u}$ (Picard linearization). Note that at this point continuity is not explicitly enforced, meaning that $\hat{\mathbf{u}}^{n+1}$ will generally not be solenoidal. It should also be noted that the same differential operator is applied to all components of $\hat{\mathbf{u}}^{n+1}$, and that consequently only one system with N_d right-hand sides needs to be solved, providing additional efficiency.

The second phase – an original contribution of [83] – computes $\mathbf{u}$ as a correction of the intermediate velocity obtained from (2.18). It can be seen as the projection of $\hat{\mathbf{u}}^{n+1}$ onto a divergence-free space. The div-curl operator present in (2.19) lends its name to this step.

The final phase consists of solving a diffusion equation, commonly referred to as the pressure Poisson problem, for p^{n+1}. The equation (2.20a) is obtained by taking the divergence of the momentum equation and simplifying using the incompressibility constraint.

We note for clarity that the colon operator present in (2.20a) denotes the Frobenius inner product, i.e., $\mathbf{A} : \mathbf{B} = \mathrm{tr}\left(\mathbf{A}^T\mathbf{B}\right)$. The boundary condition (2.20b) is formulated by taking the inner product of the momentum equation with the boundary normal, additionally substituting the term $\Delta\mathbf{u}^{n+1}$ with $-\nabla \times (\nabla \times \mathbf{u}^{n+1})$. This rotational form is based on the identity (2.12), and is commonly used in order to improve mass conservation at the boundary Γ_D. It should be emphasized that the second derivatives of the velocity do not need to computed in order to evaluate the right-hand side of (2.20b). Recall that the advection-diffusion step is solved using the equivalent first-order system, and hence the entries of the tensor $\nabla\hat{\mathbf{u}}^{n+1}$ are directly available as the auxiliary variables. Consequently, the vorticity approximation $\hat{\boldsymbol{\omega}}^{n+1}$, and therefore by virtue of (2.19a) also $\boldsymbol{\omega}^{n+1}$, can be computed without differentiation. The second boundary condition (2.20c) is obtained by taking the inner product of the open boundary condition (2.9b) with $\mathbf{n}$.

A keen reader will notice the addition of the term $-\nu\nabla \cdot \mathbf{u}^{n+1}$ on the right-hand side of (2.20c). In standard pressure correction-type schemes, this term is included to prevent numerical locking, a phenomenon wherein the pressure and the normal derivative of the velocity are fixed at Γ_O and cannot evolve in time. This erroneous behavior appears when the condition imposed on the open boundary in the pressure Poisson problem is formulated directly using the velocity computed in the advection-diffusion step. Indeed, considering (2.18c) together with (2.20c), it is readily apparent that using $\hat{\mathbf{u}}^{n+1}$ instead of $\mathbf{u}^{n+1}$ in the latter condition would result in $p^0|_{\Gamma_O} = \ldots = p^{n+1}|_{\Gamma_O}$. An insightful discussion of numerical locking can be found in [24]. In the strategy outlined above, thanks to the presence of the second step, this phenomenon does not occur. Nevertheless, it was found that the addition of the aforementioned term improved the time convergence properties of the scheme. The reader may find an in-depth analysis of open boundary conditions in pressure-correction-type schemes in [63].

The final issue which remains to be addressed is initialization. Since the BDF schemes are not self-starting, if $J > 1$ then the first J time iteration must use appropriately lower orders for the time discretization and extrapolation. Furthermore, even for $J = 1$, the initial guess for the pressure p^0 must be properly constructed, as it is not part of the problem statement formulated above (unlike $\mathbf{u}^0$, which is supplied by the initial condition). This topic is comprehensively discussed in [10]. In the examples considered within the present work, p^0 is either known (as is the case in the convergence studies presented in

chapter 6) or can be assumed to be trivial (when the flow is initially stationary, it is natural to take $p^0 \equiv 0$).

Chapter 3

The Least-Squares Spectral/*hp* Element Method

This chapter contains a review of the least-squares spectral/hp element method. First, based on [40, 52], we discuss the least-squares formulation, which allows for the discretization of a set of partial differential equations, giving rise to a system linear algebraic equations. Next, we focus on the construction of the basis functions used in the finite element approximation, and show how to leverage the mapping between reference and physical space to handle non-trivial geometries. Finally, we review how to numerically evaluate domain and boundary integrals arising from the discrete formulation. The reader can find a fuller explanation these concepts, foundational to the finite element method, in textbooks such as [110, 111, 112].

3.1 The Least-Squares Formulation

Let $\Omega \subset \mathbb{R}^{N_d}$ be an open bounded set, $\partial\Omega$ be its closure, and $\bar{\Omega} = \Omega \cup \partial\Omega$, where $N_d \in \{2, 3\}$ is the spatial dimension. We consider the abstract boundary-value problem

$$\mathscr{A}(\mathbf{u}) = \mathbf{f} \qquad \text{in } \Omega, \tag{3.1a}$$

$$\mathscr{B}(\mathbf{u}) = \mathbf{g} \qquad \text{on } \partial\Omega, \tag{3.1b}$$

where $\mathscr{A}$ and $\mathscr{B}$ are linear, first-order partial differential operators defined in the domain and on the boundary, respectively, $\mathbf{u} \in \mathscr{V}$ denotes the dependent variable, $\mathscr{V}$ being the space of admissible solutions, and $\mathbf{f} \in [L_2(\Omega)]^{N_e}$ and $\mathbf{g} \in [L_2(\partial\Omega)]^{N_{eb}}$ represent the forcing

terms. We assume that the problem is well-posed, and that the solution is sufficiently regular, e.g., $\mathscr{V} = \left[H^1\left(\bar{\Omega}\right)\right]^{N_u}$. N_u, N_e, and N_{eb} denote the number of unknowns, equations defined in the domain, and equations defined on the boundary, respectively. N_u and N_e may not be equal for over-determined problems. The requirements imposed on the operators imply that systems not conforming to this form must first be converted to the first order and linearized. A robust investigation of the latter issue, namely whether the operators should be linearized before or after the least-squares problem is posed, can be found in [80]. Furthermore, the lack of time-dependent terms in (3.1) indicates that the governing PDEs have been discretized in time. This is common practice in most numerical methods, as space-time coupled formulations typically carry an infeasible computational cost, particularly for three-dimensional problems. We now introduce the least-squares functional $\mathscr{J} : \mathscr{V} \to \mathbb{R}$, which is central to the least-squares method. It is formed using the norms of the residuals of the system (3.1),

$$\mathscr{J}\left(\mathbf{u}; \mathbf{f}, \mathbf{g}\right) = \frac{1}{2}\left(\left\|\mathscr{A}\left(\mathbf{u}\right) - \mathbf{f}\right\|_{\Omega,0}^2 + \left\|\mathscr{B}\left(\mathbf{u}\right) - \mathbf{g}\right\|_{\partial\Omega,0}^2\right). \tag{3.2}$$

Clearly, if (3.1) possesses a unique solution $\mathbf{u}$, then $\mathbf{u}$ is the strict global minimum of $\mathscr{J}$. The inverse is also true. This observation leads to the least-squares principle, which can be postulated as the following minimization problem:

$$\min_{\tilde{\mathbf{u}} \in \mathscr{V}} \mathscr{J}\left(\tilde{\mathbf{u}}; \mathbf{f}, \mathbf{g}\right). \tag{3.3}$$

The Euler-Lagrange equations associated with (3.3) can be expressed in the following variational form [81]: find $\mathbf{u} \in \mathscr{V}$, such that for every $\boldsymbol{v} \in \mathscr{V}$,

$$\mathscr{L}\left(\mathbf{u}, \boldsymbol{v}\right) = \mathscr{F}\left(\boldsymbol{v}\right), \tag{3.4}$$

where the bilinear form $\mathscr{L}$ and functional $\mathscr{F}$ are given by

$$\mathscr{L}\left(\mathbf{u}, \boldsymbol{v}\right) = \left(\mathscr{A}\left(\boldsymbol{v}\right), \mathscr{A}\left(\mathbf{u}\right)\right)_{\Omega,0} + \left(\mathscr{B}\left(\boldsymbol{v}\right), \mathscr{B}\left(\mathbf{u}\right)\right)_{\partial\Omega,0}, \tag{3.5a}$$

$$\mathscr{F}\left(\boldsymbol{v}\right) = \left(\mathscr{A}\left(\boldsymbol{v}\right), \mathbf{f}\right)_{\Omega,0} + \left(\mathscr{B}\left(\boldsymbol{v}\right), \mathbf{g}\right)_{\partial\Omega,0}. \tag{3.5b}$$

The discrete least-squares formulation results from restricting $\mathscr{V}$ to a finite-dimensional subspace $\mathscr{V}_{hp}$. As there are no compatibility conditions imposed on the choice of this subspace, we are free to use the same space for each of the components of the vector-valued function $\mathbf{u}$. $\mathscr{V}_{hp}$ can therefore be expressed as

$$\mathscr{V} \supset \mathscr{V}_{hp} = \operatorname{span}\left\{\mathbf{1}_{N_u} v_1, \mathbf{1}_{N_u} v_2, \ldots, \mathbf{1}_{N_u} v_{N_g}\right\}, \tag{3.6}$$

where $v_i \in H^1\left(\bar{\Omega}\right)$ are scalar-valued functions, N_g is the dimension of $\mathscr{V}_{hp}$, and $\mathbf{1}_{N_u}$ denotes the vector of ones of length N_u. Thanks to (3.6), any field $\mathbf{u} \in \mathscr{V}_{hp}$ can be expressed in terms of v_i,

$$\mathbf{u} = \sum_{i=1}^{N_g} v_i \hat{\mathbf{u}}_i, \tag{3.7}$$

where $\hat{\mathbf{u}}_i \in \mathbb{R}^{N_u}$, $i = 1, \ldots, N_g$ are vectors of weights. The statement of the discrete least-squares principle reads as follows: find $\mathbf{u} \in \mathscr{V}_{hp}$, which satisfies (3.4) for every $\boldsymbol{v} \in \mathscr{V}_{hp}$. As the form $\mathscr{L}$ is bilinear, and given (3.7), this principle leads directly to the system of algebraic equations

$$\hat{\mathbf{K}}\hat{\mathbf{u}} = \hat{\mathbf{f}}, \tag{3.8}$$

where $\hat{\mathbf{u}} = \begin{bmatrix} \hat{\mathbf{u}}_1^T & \ldots & \hat{\mathbf{u}}_{N_g}^T \end{bmatrix}^T$, and $\hat{\mathbf{K}} \in \mathbb{R}^{N_g N_u \times N_g N_u}$ and $\hat{\mathbf{f}} \in \mathbb{R}^{N_g N_u}$ are given by

$$\hat{\mathbf{K}}_{ij} = \mathscr{L}\left(\boldsymbol{v}_i, \boldsymbol{v}_j\right), \tag{3.9a}$$

$$\hat{\mathbf{f}}_i = \mathscr{F}\left(\boldsymbol{v}_i\right), \tag{3.9b}$$

where $\boldsymbol{v}_i = \mathbf{1}_{N_u} v_i$, $i = 1, \ldots, N_g$. Due to the finite element method's structural mechanics roots, the matrix $\hat{\mathbf{K}}$ is often called the stiffness matrix.

Let us now discuss the advantages and challenges of the least-squares method, as compared to its counterparts. What follows is an enumeration of the most important features of the LSFEM. A similar list can be found in the introduction of [52].

Universality and robustness. There exists a plethora of different numerical methods for the solution of PDEs, with various applicability restrictions depending on the type of equation under consideration. For example, when solving the advection-diffusion equation, standard schemes may perform poorly, or lose stability altogether, for high values of the Peclet number (dominant advection). This necessitates the use of various stabilization techniques, such as upwinding. The LSFEM will generally perform well and yield reasonably accurate results without the need for any special treatment. Additionally, as previously stated, no compatibility conditions are imposed on the choice of basis spaces. This stands in contrast to the Galerkin method, where these spaces must conform to the Ladyzhenskaya–Babuška–Brezzi (LBB) condition [32]. For the Navier-Stokes problem, this results in the velocity requiring a higher approximation order than the pressure.

Optimality. A key issue in the numerical solution of PDEs is the optimality of the method, meaning whether the order of the error is equal to the order of the approximation. The LSFEM provably meets this criterion for many types of equations. The reader can find an exhaustive analysis of the optimality of the LSFEM in various settings in [40].

Symmetric positive-definite systems. The algebraic systems resulting from the least-squares formulation are always symmetric positive-definite, even when the underlying operator is not self-adjoint, which follows immediately from (3.5). Such systems can be solved using more efficient iterative approaches, such as the conjugate gradient method.

Generality and ease of use. Since the LSFEM does not require a weak formulation, the original PDEs appear unchanged in the implementation. This allows the software to be structured so that the user only needs to provide the coefficients of the differential operator, the right-hand side, and the boundary conditions. As a result, the method becomes significantly more accessible to researchers and others without an extensive background in numerical methods or mathematics. While this benefit is difficult to quantify precisely, reducing the barrier to entry for research involving physical simulation is a valuable goal. Furthermore, the LSFEM is well-suited for solving multi-physics problems, where different PDEs govern distinct parts of the domain (e.g., fluid–structure interaction). The least-squares functional can be naturally decomposed into a sum of integrals over these subdomains, with the various operators evaluated separately and then assembled into a single, unified algebraic system.

Built-in error estimation. In order to assess the accuracy of the computed solution, a numerical method for the solution of PDEs must provide some form of error indicator. In general, constructing such an indicator may not be trivial. In case of the LSFEM, evaluating the least-squares functional immediately yields the L_2 error norm.

Higher computational cost. The most significant disadvantage of the least-squares method, compared to other kinds of FEM, is its higher computational cost. This stems chiefly from the fact that the system being solved must be of the first order. When the problem is posed in the weak form, the action of some of the differential operators may

be moved onto the trial functions using integration by parts, so that the resulting bilinear forms only contain first derivatives. Since the least-squares formulation is posed in the strong sense, the system must first be converted to the first order by introducing auxiliary unknowns, which raises the computational cost accordingly. It is sometimes argued in the literature that these additional unknowns represent physically useful quantities, such as fluxes. While this may be true, it should clearly be stated that such quantities would be vastly more efficient to compute as part of a post-processing step. Furthermore, in the context of a polynomial-based approximation, the order of the integrands present in (3.5) may be higher[1], necessitating the use of more computationally expensive integration rules.

3.2 Tensorial Basis Functions

As introduced above, the finite element discretization is based on a set of global basis functions $\Phi_i : \Omega \to \mathbb{R}$, $i = 1, \ldots, N_g$. A given field $u : \Omega \to \mathbb{R}$ can be approximated as their linear combination

$$u(\mathbf{x}) = \sum_{i=1}^{N_g} \hat{u}_i \Phi_i(\mathbf{x}) = \boldsymbol{\Phi}(\mathbf{x}) \cdot \hat{\mathbf{u}}, \tag{3.10}$$

where $\hat{\mathbf{u}} \in \mathbb{R}^{N_g}$ is the vector of their corresponding weights. Typically, these functions are piecewise polynomial, due to the ease of their numerical treatment, most notably integration and differentiation. In classical spectral methods [15], they are constructed globally, in the entire domain. However, this approach poses considerable challenges when dealing with non-trivial geometries. The finite element method circumvents these by introducing a domain decomposition into small, non-overlapping, geometrically simple subdomains – the eponymous elements. The basis is then constructed locally, at the element level, with the global modes composed of individual element contributions, as depicted in Figure 3.1. This is done in conformance with the regularity constraints imposed on the global bases by the underlying mathematical formulation [17, 18]. Methods which aim to achieve convergence by reducing the element size are known as h-type methods (h being the mesh size). Conversely, p-type methods leave the mesh size fixed, and increase the order of the

[1] In the LSFEM, this order is double that of $\mathscr{A}(\mathbf{u})$. In the Galerkin method, it is equal to the order of $\mathscr{A}(\mathbf{u})$ plus the order of $\mathbf{u}$. Since $\mathscr{A}$ may involve products of multiple fields governed by the approximation (e.g. in the case of the advection-diffusion equation), this is potentially disadvantageous.

approximation p. The spectral/hp method takes advantage of both these strategies.

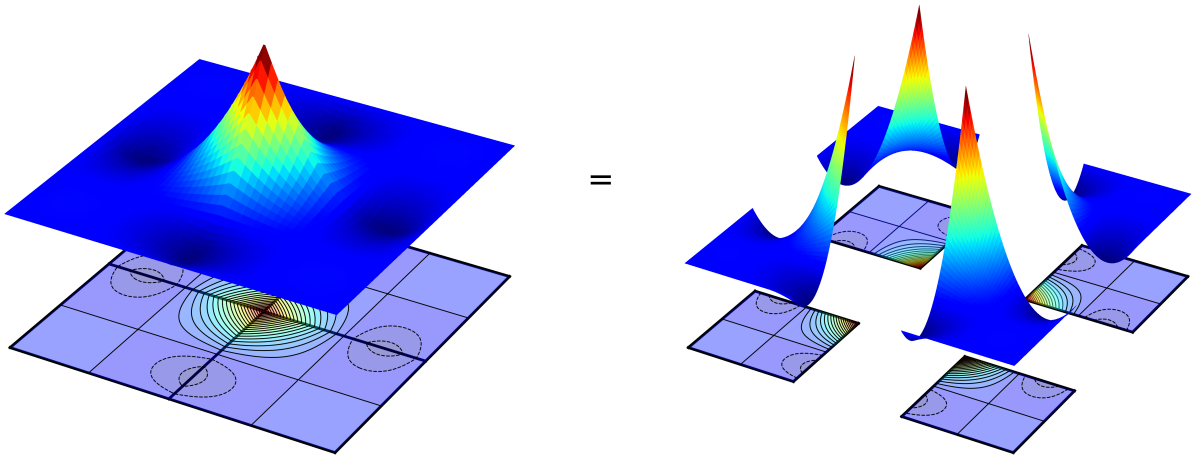


Figure 3.1. Decomposition of a global basis mode into its individual element contributions. Note the discontinuity of the derivative across element boundaries.

The scope of the present work is limited only to the Lagrange nodal basis. First, let us focus on the one-dimensional case. Let $\Omega_r^1 = [-1, 1]$ be the reference line element, and let $\mathcal{N}$ be the set of node coordinates ξ_i, $i = 1, \ldots, p+1$, such that $\xi_i \in \Omega_r^1$. The Lagrange nodal basis functions are the set of the $p+1$ polynomials ψ_i of order p, such that $\psi_i(\xi_j) = \delta_{ij}$:

$$\psi_i(\xi) = \ell_i^{\mathcal{N}}(\xi) = \prod_{\substack{j=1 \\ j \neq i}}^{p+1} \frac{\xi - \xi_j}{\xi_i - \xi_j}. \tag{3.11}$$

This approach allows for a very intuitive interpretation of the weights $\hat{u}_i$, which represent the values of the approximation at their corresponding nodes. By virtue of this property, including in the set of nodes the endpoints $\{-1, 1\}$, which are shared with the neighboring elements, yields an approximation which is globally continuous. Additionally, the coordinates of the internal nodes must be selected appropriately. Spacing them uniformly gives rise to the Runge effect, meaning that the resulting Lagrange polynomials exhibit highly oscillatory behavior, degrading the quality of the approximation. A popular choice which alleviates this problem is to place the nodes at the zeros of the Lobatto polynomial [99] of order $p - 1$:

$$\mathcal{N} = \{\xi : P_p'(\xi) = 0\} \cup \{-1, 1\}, \tag{3.12}$$

where P_p denotes the Legendre polynomial of order p (the Lobatto polynomial is its derivative). In the context of the Galerkin method, such a distribution can also be beneficial to

the structure of the mass matrix $m_{ij} = (\Phi_i, \Phi_j)_\Omega$, however, in the least-squares method, this matrix bears no special significance. A robust discussion of the selection of the expansion, including, notably, modal bases, can be found in [54]. It is worth emphasizing that while only the basis described by (3.11) and (3.12) is considered herein, the extension of the concepts described in subsequent chapters to other approximations is straightforward.

The basis functions in the reference quadrilateral element $\Omega_r^2 = [-1,1] \times [-1,1]$ can be constructed using a tensorial expansion of the one-dimensional basis (3.11). In the case of the nodal basis, this can be interpreted as arranging the nodes constituting $\mathcal{N}$ along a Cartesian grid, with the grid coordinates in each direction determined by (3.11). The quantity $u : \Omega_r^2 \to \mathbb{R}$ can then be approximated as

$$u(\xi,\eta) = \sum_{i=1}^{p+1}\sum_{j=1}^{p+1} \psi_i(\xi)\psi_j(\eta)\hat{u}_{ij} = \boldsymbol{\varphi}_e(\xi,\eta)\cdot\hat{\mathbf{u}}, \tag{3.13}$$

where $\boldsymbol{\varphi}_e : \Omega_r^2 \to \mathbb{R}^{|\mathcal{N}|}$ is the vector of basis functions, and $\hat{\mathbf{u}} \in \mathbb{R}^{|\mathcal{N}|}$ the vector of their corresponding weights. Care must be taken when ordering the elements of $\boldsymbol{\varphi}_e$, so that they conform to the grid structure, as indicated by the double-indexed notation $\hat{u}_{ij}$. An example for $p = 4$ has been shown in Figure 3.2. The basis in the hexahedral element $\Omega_r^3 = [-1,1] \times [-1,1] \times [-1,1]$ can be constructed similarly to (3.13):

$$u(\xi,\eta,\zeta) = \sum_{i=1}^{p+1}\sum_{j=1}^{p+1}\sum_{k=1}^{p+1} \psi_i(\xi)\psi_j(\eta)\psi_k(\zeta)\hat{u}_{ijk} = \boldsymbol{\varphi}_e(\xi,\eta,\zeta)\cdot\hat{\mathbf{u}}. \tag{3.14}$$

It should be noted that the expansions (3.13) and (3.14) have been given under the assumption that the same 1D basis has been employed along every axis. It is possible to use different approximation orders, or even altogether different basis types, in different directions. This may be beneficial if the underlying physical problem exhibits preferential behavior along one axis. An example of such a case can be found in [33]. The scope of the present work is limited to isotropic quadrilateral and hexahedral elements, however, the algorithms presented in chapter 4 can easily be generalized to any multidimensional basis possessing a tensor-product structure. For information on constructing such bases in triangular and tetrahedral elements, the reader may refer to [11, 13, 54, 107].

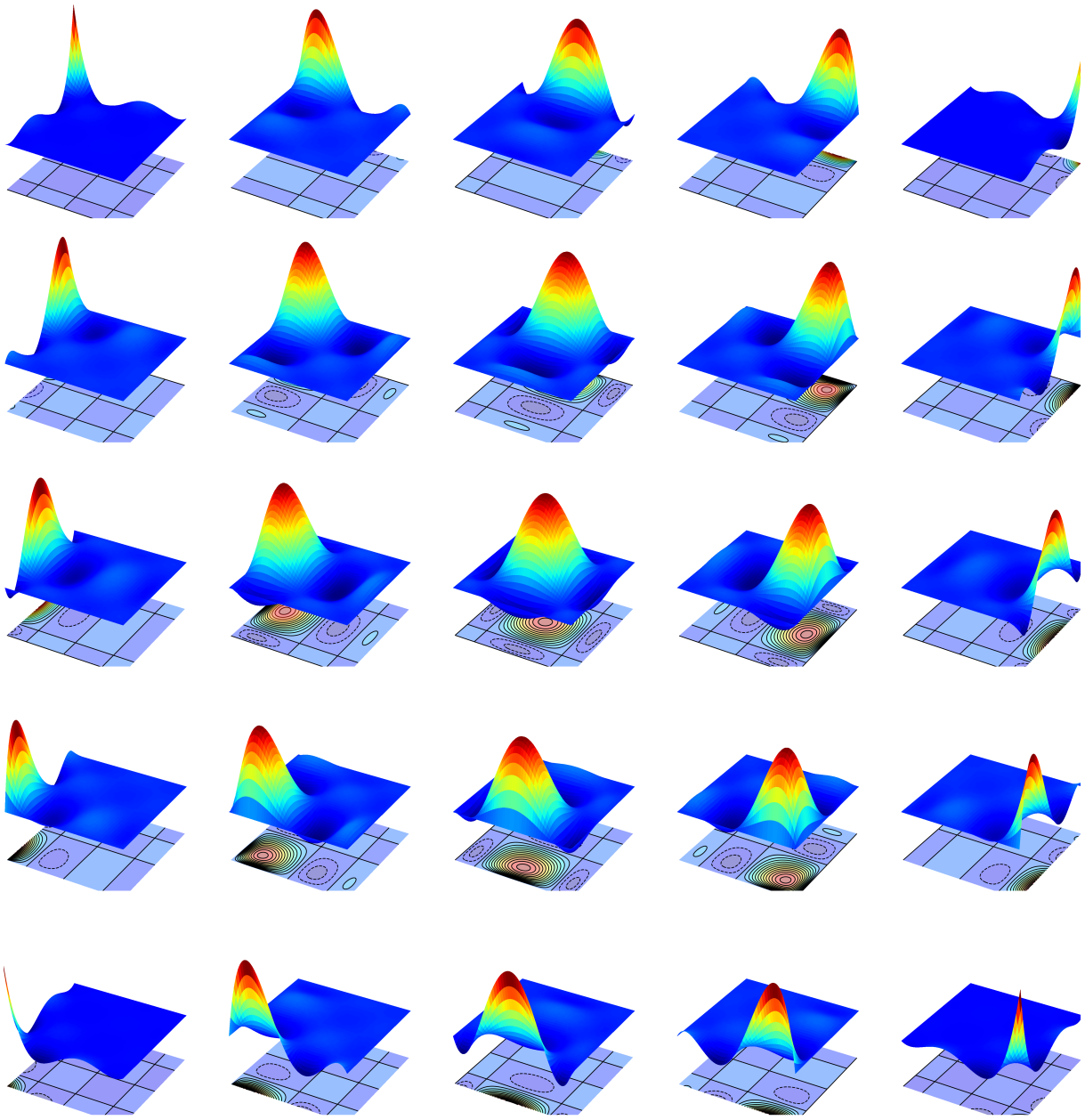


Figure 3.2. C^0 Lagrange nodal basis functions in reference quadrilateral element, $p = 4$.

3.3 Coordinate Transformation

The construction of the bases in the previous section was performed in a reference (standard) region. In order for the finite element method to be practically applicable, the global approximations must be able to represent any domain geometry. To achieve this, Ω is first decomposed into a tesselation of $|\mathcal{E}|$ elements, such that

$$\Omega = \bigcup_{e \in \mathcal{E}} \Omega_e \wedge \forall_{i,j \in \mathcal{E}, i \neq j} \, \Omega_i \cap \Omega_j = \emptyset. \tag{3.15}$$

In each element, a bijective mapping $\boldsymbol{\chi}_e : \Omega_r \to \Omega_e$ between reference and physical coordinates is defined, as illustrated in Figure 3.3. Analogously to the finite element discretization, this mapping is also realized using a set of basis functions, often referred to as shape functions. Let $\{s_i : \Omega_r \to \mathbb{R},\, i \in \mathcal{S}\}$ be the set of shape functions, and $\mathbf{v}_i^e \in \mathbb{R}^{N_d}$ their corresponding weights. $\boldsymbol{\chi}_e$ can be expressed as

$$\boldsymbol{\chi}_e(\boldsymbol{\xi}) = \sum_{i \in \mathcal{S}} \mathbf{v}_i^e s_i(\boldsymbol{\xi}) = \mathbf{V}_e \mathbf{s}(\boldsymbol{\xi}), \tag{3.16}$$

where the matrix $\mathbf{V}_e \in \mathbb{R}^{N_d \times |\mathcal{S}|}$ has been formed in such a way that $\mathbf{v}_i^e$ constitutes its i-th column. The nodal Lagrange expansion is a convenient choice for the geometric basis. Let g denote its order, independent of the approximation order p. For $g = 1$, the vectors $\mathbf{v}_i^e$ simply describe the positions of the element's vertices in physical space. Setting $g > 1$ leads to curvilinear elements, which are able to more accurately represent curved geometric features. Depending on whether g is less than, equal to, or greater than p, the element is called subparametric, isoparametric, or superparametric, respectively [25].

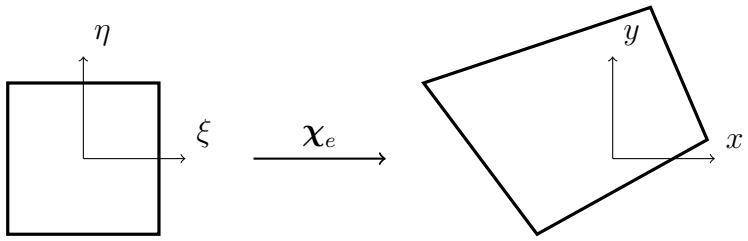


Figure 3.3. Coordinate transformation of a quadrilateral element from reference to physical space.

The mapping $\boldsymbol{\chi}_e$ can now be used to construct the approximation basis in physical space. Let $\boldsymbol{\phi}_e : \Omega_e \to \mathbb{R}^{|\mathcal{N}|}$ denote the vector of these bases. The value of $\boldsymbol{\phi}_e$ at the point $\mathbf{x} \in \Omega_e$ is equal to the value of $\boldsymbol{\varphi}_e$ at the point $\boldsymbol{\xi} \in \Omega_r$, such that $\mathbf{x} = \boldsymbol{\chi}_e(\boldsymbol{\xi})$. Written

concisely,

$$\forall_{\boldsymbol{\xi}\in\Omega_r}\, \boldsymbol{\phi}_e\left(\boldsymbol{\chi}_e\left(\boldsymbol{\xi}\right)\right) = \boldsymbol{\varphi}_e\left(\boldsymbol{\xi}\right), \tag{3.17}$$

leading to the approximation

$$u(\mathbf{x}) = \boldsymbol{\phi}_e(\mathbf{x}) \cdot \hat{\mathbf{u}} = \boldsymbol{\phi}_e\left(\boldsymbol{\chi}_e(\boldsymbol{\xi})\right) \cdot \hat{\mathbf{u}} = \boldsymbol{\varphi}_e\left(\boldsymbol{\xi}\right) \cdot \hat{\mathbf{u}}. \tag{3.18}$$

The Jacobian matrix $\mathbf{J}_e : \Omega_r \to \mathbb{R}^{N_d \times N_d}$ of the coordinate transformation is given by

$$\left(j_e\right)_{ij} = \frac{\partial \chi_i}{\partial \xi_j} = \sum_{k\in\mathcal{S}} v_{ik}^e \frac{\partial s_k}{\partial \xi_j}, \tag{3.19}$$

In matrix notation, (3.19) can be expressed as

$$\mathbf{J}_e = \frac{\partial \boldsymbol{\chi}_e}{\partial \boldsymbol{\xi}} = \mathbf{V}_e \begin{bmatrix} \frac{\partial \mathbf{s}}{\partial \xi_1} & \dots & \frac{\partial \mathbf{s}}{\partial \xi_{N_d}} \end{bmatrix} = \mathbf{V}_e \nabla_\xi \mathbf{s}. \tag{3.20}$$

Using the chain rule, we can formulate the derivative in reference space in terms of the physical derivative:

$$\frac{\partial u}{\partial \xi_i} = \sum_{j=1}^{N_d} \frac{\partial u}{\partial x_j} \frac{\partial x_j}{\partial \xi_i}. \tag{3.21}$$

From the definition of the Jacobian (3.19), it follows that

$$\nabla_\xi u = \mathbf{J}_e^T \nabla_x u. \tag{3.22}$$

For the Jacobian matrix to be invertible in the entire region where it is defined, the mapping $\boldsymbol{\chi}_e$ must satisfy certain conditions, depending on the element type. For example, in the case of quadrilateral elements, it must produce a convex shape (i.e. the internal angles must be less than 180°). Hexahedral elements in general do not offer such a straightforward criterion, although some algorithms for efficiently checking the sign of the Jacobian determinant exist [57]. In the present work, we assume that $\det \mathbf{J}_e > 0$ everywhere, and therefore

$$\nabla_x u = \mathbf{J}_e^{-T} \nabla_\xi u = \mathbf{J}_e^{-T} \nabla_\xi^T \boldsymbol{\varphi}_e \hat{\mathbf{u}}. \tag{3.23}$$

The equations (3.18) and (3.23) are very important to any type of finite element method, as they allow us to reason about quantities in the physical space using only the bases defined in the reference region.

The final issue relating to coordinate transformation that must be addressed is the calculation of boundary normals. These may be needed, for example, to formulate boundary

conditions, or to compute fluxes of various quantities passing through a boundary. First, let us consider the reference quadrilateral element. Each of its edges can be parametrized in the bi-unit segment, using the transformation $\boldsymbol{\lambda} : [-1, 1] \to \Omega_r^2$, consisting only of a rotation and a translation

$$\boldsymbol{\lambda}(\xi) = \mathbf{R}_f \begin{bmatrix} \xi \\ 0 \end{bmatrix} + \mathbf{t}_f, \tag{3.24}$$

where the rotation matrix $\mathbf{R}_f \in \mathbb{R}^{N_d \times N_d}$ and translation vector $\mathbf{t}_f \in \mathbb{R}^{N_d}$ depend on the specific edge being considered. We further assume that the parametrization $\boldsymbol{\lambda}$ exhibits a counter-clockwise orientation with regard to the reference element boundary. Applying $\boldsymbol{\chi}_e$ allows the edge in *physical* space to be parametrized by $\boldsymbol{\ell} : [-1, 1] \to \Omega_e$, such that

$$\boldsymbol{\ell}(\xi) = \boldsymbol{\chi}_e(\boldsymbol{\lambda}(\xi)). \tag{3.25}$$

The tangent vector $\boldsymbol{\tau}$ (in physical space) is given by

$$\boldsymbol{\tau}(\xi) = \frac{\mathrm{d}\boldsymbol{\ell}}{\mathrm{d}\xi} = \frac{\partial \boldsymbol{\chi}_e}{\partial \boldsymbol{\lambda}} \frac{\partial \boldsymbol{\lambda}}{\partial \xi_1} = \mathbf{J}_e(\boldsymbol{\lambda}(\xi)) \mathbf{R}_f \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{3.26}$$

Because we assumed a counter-clockwise orientation, the outward-pointing normal can now be obtained by rotating the tangent clockwise by $\frac{\pi}{2}$, leading to

$$\mathbf{n}(\xi) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \boldsymbol{\tau}(\xi) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{J}_e(\boldsymbol{\lambda}(\xi)) \mathbf{R}_f \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{3.27}$$

Since $\boldsymbol{\lambda}$ involves a rotation by a multiple of $\frac{\pi}{2}$, the entries of $\mathbf{R}_f$ must belong to the set $\{-1, 0, 1\}$. Therefore, the equation (3.27) can further be simplified so that $\mathbf{n}$ consists of (possibly negated) entries of the Jacobian matrix.

The computation of the boundary normal in hexahedral elements follows a similar process. Each of the faces of the reference hexahedron can be parametrized in the bi-unit square, using the transformation $\boldsymbol{\sigma} : [-1, 1] \times [-1, 1] \to \Omega_r^3$,

$$\boldsymbol{\sigma}(\xi, \eta) = \mathbf{R}_f \begin{bmatrix} \xi \\ \eta \\ 0 \end{bmatrix} + \mathbf{t}_f. \tag{3.28}$$

In this case, we require that $\boldsymbol{\sigma}$ be defined so that $\boldsymbol{\sigma}(1, 0) \times \boldsymbol{\sigma}(0, 1)$ points outward relative to the boundary of Ω_r^3. The parametrization in physical space $\tilde{\mathbf{s}} : [-1, 1] \times [-1, 1] \to \Omega_e$ is given by

$$\tilde{\mathbf{s}}(\xi, \eta) = \boldsymbol{\chi}_e(\boldsymbol{\sigma}(\xi, \eta)). \tag{3.29}$$

Note that the notation $\tilde{\mathbf{s}}$ has been introduced to avoid confusion with the vector of shape functions $\mathbf{s}$. Fixing either of the coordinates in the mapping (3.29) at some point $\boldsymbol{\xi}$ defines two curves on the face of the physical element, both passing through $\boldsymbol{\xi}$. Since $\boldsymbol{\chi}_e$ and $\boldsymbol{\sigma}$ are non-singular, these curves will not be parallel at $\boldsymbol{\xi}$. Therefore, their tangents $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$, which are, by construction, also tangent to the element surface, will not coincide at $\boldsymbol{\xi}$. The normal can therefore be defined as the cross-product of $\boldsymbol{\tau}_1$ and $\boldsymbol{\tau}_2$. This reasoning can be expressed as

$$\boldsymbol{\tau}_1(\xi,\eta) = \frac{\partial \tilde{\mathbf{s}}}{\partial \xi} = \mathbf{J}_e\left(\boldsymbol{\sigma}\left(\xi,\eta\right)\right)\mathbf{R}_f \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \tag{3.30a}$$

$$\boldsymbol{\tau}_1(\xi,\eta) = \frac{\partial \tilde{\mathbf{s}}}{\partial \eta} = \mathbf{J}_e\left(\boldsymbol{\sigma}\left(\xi,\eta\right)\right)\mathbf{R}_f \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \tag{3.30b}$$

$$\mathbf{n}(\xi,\eta) = \boldsymbol{\tau}_1(\xi,\eta) \times \boldsymbol{\tau}_2(\xi,\eta). \tag{3.30c}$$

Due to $\mathbf{R}_f$ being composed of elemental rotations by multiples of $\frac{\pi}{2}$, (3.30c) simplifies to the cross-product of rows of the Jacobian matrix.

3.4 Integrals

Integration is one of the key operations one must be able to perform in the context of the finite element approximation, e.g., to evaluate (3.9). Since any integral over the domain Ω can be decomposed into the sum of integrals over the element domains Ω_e, what follows is a discussion of integration at the local (element) level. Let $f : \Omega_e \to \mathbb{R}$ be some integrable function. We now seek to evaluate $\int_{\Omega_e} f(\mathbf{x})\,\mathrm{d}\mathbf{x}$. As stated previously, it is vastly more convenient to operate in reference space. The first step is therefore to express the integral over Ω_e in terms of the integral over Ω_r. In accordance with the change of coordinates formula, we can write

$$\int_{\Omega_e} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_{\Omega_r} f\left(\boldsymbol{\chi}_e(\boldsymbol{\xi})\right) \det \mathbf{J}_e\left(\boldsymbol{\xi}\right)\,\mathrm{d}\boldsymbol{\xi} = \int_{\Omega_r} g\left(\boldsymbol{\xi}\right)\,\mathrm{d}\boldsymbol{\xi}, \tag{3.31}$$

where the function $g : \Omega_r \to \mathbb{R}$ has been introduced to make the subsequent notation more concise. To avoid having to evaluate (3.31) analytically, Gauss quadratures are ubiquitously used to numerically compute the result. The integral is approximated as a

weighted sum of the values of g at a number of points belonging to its domain. Let $\mathcal{Q}$ be the set of the quadrature points $\mathbf{q}_i \in \Omega_r$, called abscissas, and $w_i \in \mathbb{R}$ their corresponding weights. The general form of the quadrature formula is

$$\int_{\Omega_r} g(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \approx \sum_{i=1}^{|\mathcal{Q}|} w_i g(\mathbf{q}_i). \tag{3.32}$$

Let us now consider the one-dimensional reference line element Ω_r^1. There exists a large library of integration rules (abscissas and weights) one can employ in this domain [34]. Popular choices, e.g., Gauss-Legendre, Gauss-Legendre-Lobatto, and Gauss-Radau, are based on the distribution of abscissas at the zeros of families of orthogonal polynomials. In the Galerkin method, for certain problems, pairing the right bases with the appropriate quadrature leads to a more sparse structure of the resulting algebraic systems. For example, combining the nodal Lobatto basis with the Gauss-Lobatto quadrature of the appropriate order results in the mass matrix being diagonal. In the least-squares method, this is not the case. The only aspect which needs to be considered is the accuracy of the integration rule. The clear choice is therefore the Gauss-Legendre quadrature. Let n denote its order (i.e. the number of abscissas). If the integrand is a polynomial function, this rule produces exact results for orders up to $2n-1$, compared to $2n-2$ and $2n-3$ for Gauss-Radau and Gauss-Legendre-Lobatto quadratures, respectively. The set of abscissas $\mathcal{Q}_n^1$ and corresponding weights ω_i for the Gauss-Legendre quadrature of order n are given by

$$\mathcal{Q}_n^1 = \{x \in (-1,1) : P_n(x) = 0\}, \tag{3.33a}$$

$$\omega_i = 2\left(\left(1-q_i^2\right)\left(P_n'(q_i)\right)^2\right)^{-1}. \tag{3.33b}$$

A discussion on how to accurately compute them in finite precision can be found in [53]. The number of quadrature points in one dimension $|\mathcal{Q}_n^1|$ plays an important role in this study, and will be denoted simply by q. The reader should avoid confusion with q_i, which is the coordinate of the i-th abscissa, and N_q, which denotes the total number of quadrature points in the N_d-dimensional element.

Generalizing the one-dimensional quadrature to the reference quadrilateral and hexahedral elements is straightforward, and conforms to the same tensor-product structure as higher-dimensional basis functions. Let Q_ξ and Q_η denote the order of the integration rule used in the ξ and η directions, respectively. The abscissas and weights in Ω_r^2 follow

directly from

$$\int_{\Omega_r^2} g(\xi,\eta)\,\mathrm{d}\xi\eta = \int_{-1}^{1}\int_{-1}^{1} g(\xi,\eta)\,\mathrm{d}\xi\,\mathrm{d}\eta \approx \sum_{i=1}^{Q_\xi}\sum_{j=1}^{Q_\eta} g\left(q_i,q_j\right)\omega_i\omega_j. \tag{3.34}$$

Similarly, for Ω_r^3, we have

$$\int_{\Omega_r^3} g(\xi,\eta,\zeta)\,\mathrm{d}\xi\eta\zeta \approx \sum_{i=1}^{Q_\xi}\sum_{j=1}^{Q_\eta}\sum_{k=1}^{Q_\zeta} g\left(q_i,q_j,q_k\right)\omega_i\omega_j\omega_k, \tag{3.35}$$

where Q_ζ denotes the order of the quadrature used in the ζ direction. A similar remark to the one given during the discussion of the construction of multidimensional bases can be made here. The present work is limited to isotropic elements, meaning that $Q_\xi = Q_\eta = Q_\zeta = q$. Consequently, in quadrilateral and hexahedral elements $N_q = q^{N_d}$.

The calculation of boundary integrals within the finite element approximation warrants additional attention. Analogously to the domain integrals, it is more convenient to evaluate them in a reference region, and employ a coordinate mapping to the physical boundary. The reference region for the boundary is a codimension 1 subspace of the reference region for the element. The necessary formalism describing these transformations has already been introduced during the discussion surrounding boundary normals, in equations (3.25) and (3.29). Let Λ_e be the image of an edge of the reference quadrilateral element under the mapping $\boldsymbol{\chi}_e$. The change of coordinates formula yields

$$\int_{\Lambda_e} f(\mathbf{x})\,\mathrm{d}l = \int_{\Omega_r^1} f\left(\boldsymbol{\ell}(\xi)\right)\left\|\frac{\mathrm{d}\boldsymbol{\ell}}{\mathrm{d}\xi}\right\|\,\mathrm{d}\xi. \tag{3.36}$$

Similarly, if Γ_e denotes the image of a face of the reference hexahedral element under the mapping $\boldsymbol{\chi}_e$, we have

$$\int_{\Gamma_e} f(\mathbf{x})\,\mathrm{d}s = \int_{\Omega_r^2} f\left(\tilde{\mathbf{s}}(\xi,\eta)\right)\left\|\frac{\mathrm{d}\tilde{\mathbf{s}}}{\mathrm{d}\xi}\times\frac{\mathrm{d}\tilde{\mathbf{s}}}{\mathrm{d}\eta}\right\|\,\mathrm{d}\xi\eta. \tag{3.37}$$

The determinants of these transformations can be evaluated in terms of the Jacobian matrix $\mathbf{J}_e$, as given by (3.27) and (3.30). The integrals in the N_d-1-dimensional reference space can be computed numerically, using the appropriate Gauss quadrature.

Chapter 4

Operator Evaluation Strategies

So far, we have reviewed how to discretize a field defined in the domain Ω using the spectral/hp element approximation, expressing it in terms of a finite number of parameters. We have also shown how to employ the least-squares principles to convert a set of PDEs into a system of algebraic equations, whose solution yields the parameters constituting the approximation of the unknown fields. In order to claim to be able to numerically solve the PDEs described in chapter 2, however, we must also present an efficient algorithm for solving this algebraic system. As problems arising in practical applications can be very large, it is often infeasible to solve them using direct methods, such as LU factorization, which involve $\mathcal{O}(n^3)$ operations, n being the number of unknowns. For this reason, over the past two centuries, a number of methods for the iterative solution of systems of linear equations have been developed [94]. These schemes start from an initial guess for the solution, and endeavor to improve it by repeating a sequence of algebraic operations involving the coefficient matrix and right-hand side. Krylov subspace methods have garnered the most popularity in the finite element community. Of this class, the conjugate gradient method (CGM) [43] is of particular interest to the present work. It offers excellent convergence, however it can only be used if the system of equations is symmetric positive-definite. Since the LSFEM always produces such systems, regardless of whether the partial differential operators under consideration are self-adjoint, the CGM can be used for all types of governing equations.

The CGM (as well as other Krylov subspace methods) take as their inputs the initial guess, the right-hand side vector, and the operator implied by the coefficient matrix. Crucially, only the operator is required – the individual entries of the matrix need not be

examined, only the ability to evaluate the matrix-vector product, for some given vector, is needed. This observation gives rise to different approaches for evaluating the operator, discussed in this chapter. In the global assembly approach, the matrix is computed and stored in memory. In the local element approach, the global matrix is never explicitly formed. Instead, the action of each element's contribution is evaluated locally, and the results summed. The sum-factorization technique takes a similar approach, but additionally takes advantage of the tensor-product structure of the basis and quadrature to eliminate redundant arithmetic operations. As the matrix-vector product makes up the majority of the computational effort of a CGM iteration, the selection of the operator evaluation strategy is of paramount importance to the performance of the algebraic solver. The correct choice may depend on the approximation order p, as well as the PDE under consideration. The investigation of this issue in the context of the least-squares method constitutes, to the best of the author's knowledge, a novel contribution of the present work.

Before proceeding further, for the reader's convenience, the notation describing the most important quantities which will be heavily used in the current chapter has been compiled in Table 4.1. Most of these symbols have already been defined in the preceding text, while those which were not will be properly introduced before they are used below.

Symbol	Description
p	Approximation order
q	Number of quadrature points in 1 dimension
N_d	Spatial dimension
N_u	Number of unknown scalar fields
N_e	Number of equations
N_b	Number of elemental bases, here $N_b = (p+1)^{N_d}$
N_q	Number of quadrature points in the element, here $N_q = q^{N_d}$
N_l	Number of local (elemental) DOFs, $N_l = N_b \cdot N_u$
N_f	Number of external scalar fields needed for operator evaluation

Table 4.1. Review of selected symbols.

4.1 Global Assembly

In the global assembly method, the stiffness matrix $\hat{\mathbf{K}}$ from equation (3.8) is explicitly computed. Let us show how to employ the theory developed in chapter 3 to this end. We start by discussing the fundamental issue of the structure of $\hat{\mathbf{K}}$, and the way the individual elements contribute to it. The blocks $\hat{\mathbf{K}}_{ij}$ from equation (3.9a) are defined using a bilinear form generated by an inner product (this is generally true for other types of FEM as well). Therefore, they can only have non-zero entries if the supports of the bilinear form's arguments overlap, which can be written as

$$\operatorname{supp}(\boldsymbol{v}_i) \cap \operatorname{supp}(\boldsymbol{v}_j) = \emptyset \Rightarrow \hat{\mathbf{K}}_{ij} = \mathscr{L}(\boldsymbol{v}_i, \boldsymbol{v}_j) = \mathbf{0}. \tag{4.1}$$

Since the global modes $\boldsymbol{v}_i$, $\boldsymbol{v}_j$ are constructed by adding together parts defined in individual elements, this means that $\hat{\mathbf{K}}_{ij}$ can only be non-zero if there exists an element to which both of these modes belong. This notion corresponds to the geometric adjacency relation between elements. In reasonably generated meshes, the number of an element's neighbors is bounded by a small constant, independent of the size of the mesh. The global stiffness matrix is therefore highly sparse. Furthermore, as $\mathscr{L}$ is an integral form, it can be evaluated piecewise in each element, and the contributions appropriately summed into $\hat{\mathbf{K}}$. This procedure is commonly referred to as global assembly, and has been depicted schematically in Figure 4.1. The elemental contributions can be expressed as the *local* stiffness matrix, formed by evaluating $\mathscr{L}$ using the *local* basis modes. The assembly process consists of iterating over the elements, computing the local matrix, and scattering its entries into $\hat{\mathbf{K}}$ following a local-to-global mapping. Computing the right-hand side vector $\hat{\mathbf{f}}$ can be accomplished using the same approach, evaluating $\mathscr{F}$ instead of $\mathscr{L}$.

Using (3.5), the local stiffness matrix $\mathbf{K}_e \in \mathbb{R}^{N_l \times N_l}$ and right-hand side vector $\mathbf{f}_e \in \mathbb{R}^{N_l}$ can be written as

$$(\mathbf{K}_e)_{ij} = \int_{\Omega_e} \mathscr{A}^T(\phi_i)\, \mathscr{A}(\phi_j)\, \mathrm{d}\mathbf{x} + \int_{\partial\Omega_e} \mathscr{B}^T(\phi_i)\, \mathscr{B}(\phi_j)\, \mathrm{d}\mathbf{s} = (\mathbf{M}_A)_{ij} + (\mathbf{M}_B)_{ij}\,, \tag{4.2a}$$

$$(\mathbf{f}_e)_i = \int_{\Omega_e} \mathscr{A}^T(\phi_i)\, \mathbf{f}(\mathbf{x})\, \mathrm{d}\mathbf{x} + \int_{\partial\Omega_e} \mathscr{B}^T(\phi_i)\, \mathbf{g}(\mathbf{x})\, \mathrm{d}\mathbf{s} = (\mathbf{f}_A)_i + (\mathbf{f}_B)_i\,, \tag{4.2b}$$

where $N_l = N_b \cdot N_u$ is the number of local degrees of freedom (DOFs), and $\partial\Omega_e$ is the (possibly empty) intersection of the element boundary with $\partial\Omega$. The matrices $\mathbf{M}_A$ and $\mathbf{M}_B$, denoting the domain and boundary integrals, respectively, have been introduced to

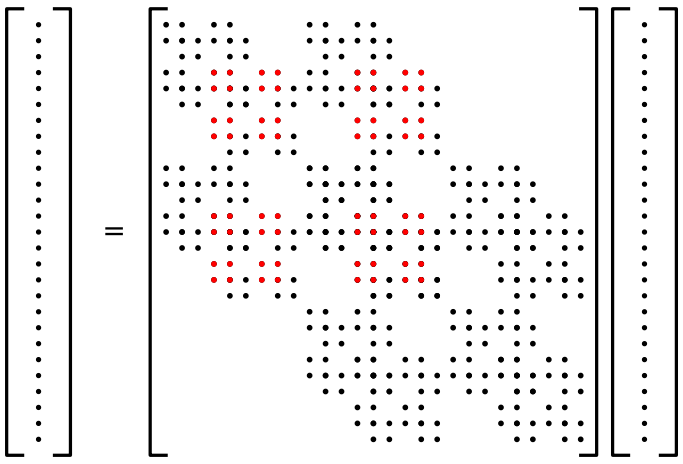


Figure 4.1. Graphical representation of the global assembly approach. Red entries indicate the contributions of a single element.

make the subsequent notation more concise, as have the vectors $\mathbf{f}_A$ and $\mathbf{f}_B$. Any first-order linear operators $\mathscr{A}$ and $\mathscr{B}$ can be expressed as

$$\mathscr{A}(\mathbf{v}) = \mathbf{A}_0\mathbf{v} + \sum_{i=1}^{N_d} \mathbf{A}_i \frac{\partial \mathbf{v}}{\partial x_i}, \tag{4.3a}$$

$$\mathscr{B}(\mathbf{v}) = \mathbf{B}_0\mathbf{v} + \sum_{i=1}^{N_d} \mathbf{B}_i \frac{\partial \mathbf{v}}{\partial x_i}, \tag{4.3b}$$

where $\mathbf{A}_i : \Omega \to \mathbb{R}^{N_e \times N_u}$, $\mathbf{B}_i : \Omega \to \mathbb{R}^{N_{eb} \times N_u}$, $i = 0, \ldots, N_d$ are matrix-valued functions. Utilizing (3.31) and (4.3a) results in $\mathbf{M}_A$ taking the form

$$(\mathbf{M}_A)_{ij} = \int_{\Omega_r} \left(\mathbf{A}_0\phi_i + \sum_{k=1}^{N_d} \mathbf{A}_k \frac{\partial \phi_i}{\partial x_k} \right)^T \left(\mathbf{A}_0\phi_j + \sum_{k=1}^{N_d} \mathbf{A}_k \frac{\partial \phi_j}{\partial x_k} \right) \det \mathbf{J}_e \, \mathrm{d}\boldsymbol{\xi}. \tag{4.4}$$

Recall that the integral present in (4.4) will be evaluated using a Gauss quadrature rule, involving N_q abscissas $\mathbf{q}_i$ (defined in reference space) and their corresponding weights w_i. We introduce the auxiliary matrices $\mathbf{N}_i \in \mathbb{R}^{N_b \times N_q}$, $i = 0, \ldots, N_d$, containing the values of the bases and their physical derivatives at the quadrature points:

$$(n_0)_{ij} = \phi_i\left(\boldsymbol{\chi}_e\left(\mathbf{q}_j\right)\right) = \varphi_i\left(\mathbf{q}_j\right), \tag{4.5a}$$

$$(n_k)_{ij} = \frac{\partial \phi_i}{\partial x_k}\left(\mathbf{q}_j\right) = \left(\mathbf{J}_e^{-T}\nabla_\xi \varphi_i\right)_k\left(\mathbf{q}_j\right), \; k = 1, \ldots, N_d. \tag{4.5b}$$

Let $\boldsymbol{\kappa}_i \in \mathbb{R}^{N_q N_e \times N_b N_u}$, $i = 0, \ldots, N_d$ describe the action of the individual components of the partial differential operator at quadrature points, expressed as

$$(\boldsymbol{\kappa}_k)_{ij} = \mathbf{A}_k\left(\boldsymbol{\chi}_e\left(\mathbf{q}_i\right)\right)\left(N_k\right)_{ji}. \tag{4.6}$$

The total action of $\mathscr{A}$, denoted as $\mathbf{H}_A \in \mathbb{R}^{N_q N_e \times N_b N_u}$, is given by their sum:

$$\mathbf{H}_A = \sum_{k=0}^{N_d} \boldsymbol{\kappa}_i. \tag{4.7}$$

Similarly, the value of the forcing terms at quadrature points can be expressed as the vector $\mathbf{f}_q \in \mathbb{R}^{N_q N_e}$, defined as

$$(\mathbf{f}_q)_i = \mathbf{f}\left(\boldsymbol{\chi}_e\left(\mathbf{q}_i\right)\right). \tag{4.8}$$

Let $\mathbf{W}_A \in \mathbb{R}^{N_q N_e \times N_q N_e}$ be a diagonal matrix formed using the quadrature weights and values of the Jacobian determinant at their corresponding abscissas, such that

$$(\mathbf{W}_A)_{ii} = \operatorname{diag}\left(\mathbf{1}_{N_e} w_i \det \mathbf{J}_e\left(\mathbf{q}_i\right)\right). \tag{4.9}$$

Employing (3.32), (4.4), (4.7), (4.8), and (4.9) together, the parts of the local system pertaining to the domain integral may be formulated as

$$\mathbf{M}_A = \mathbf{H}_A^T \mathbf{W}_A \mathbf{H}_A, \tag{4.10a}$$

$$\mathbf{f}_A = \mathbf{H}_A^T \mathbf{W}_A \mathbf{f}_q. \tag{4.10b}$$

Following a very similar process, the boundary terms for each edge or face can be expressed as

$$\mathbf{M}_B^f = \mathbf{H}_B^T \mathbf{W}_B \mathbf{H}_B, \tag{4.11a}$$

$$\mathbf{f}_B^f = \mathbf{H}_B^T \mathbf{W}_B \mathbf{g}_q, \tag{4.11b}$$

where the matrices $\mathbf{H}_B$, $\mathbf{W}_B$, and vector $\mathbf{g}_q$ correspond to $\mathbf{H}_A$, $\mathbf{W}_A$, and $\mathbf{f}_q$, respectively. The index f has been used to indicate that (4.11) needs to be evaluated separately for each edge or face of the element which coincides with $\partial\Omega$, and the results summed. The only differences in reasoning, as compared to calculating the domain terms, other than the obvious issue of substituting $\mathbf{B}_i$ for $\mathbf{A}_i$ and $\mathbf{g}$ for $\mathbf{f}$, lie in the choice of quadrature rule, which must be defined in the boundary domain, and the determinant of the geometric transformation, which is given by (3.36) or (3.37), depending on the element type.

4.1.1 Static condensation

Static condensation is a technique which can be used to reduce the size of the global system of equations using only element-level operations, first introduced in [41]. Let us

consider the algebraic system $\mathbf{K}\mathbf{x} = \mathbf{f}$. We introduce the following, for the time being, arbitrary, block decomposition

$$\mathbf{K}\mathbf{x} = \begin{bmatrix} \mathbf{K}_S & \mathbf{K}_U \\ \mathbf{K}_L & \mathbf{K}_D \end{bmatrix} \begin{bmatrix} \mathbf{x}_S \\ \mathbf{x}_D \end{bmatrix} = \begin{bmatrix} \mathbf{f}_S \\ \mathbf{f}_D \end{bmatrix} = \mathbf{f}. \tag{4.12}$$

Assuming that $\mathbf{K}_D$ is invertible, the second equation can be rewritten as

$$\mathbf{x}_D = \mathbf{K}_D^{-1} \left(\mathbf{f}_D - \mathbf{K}_L \mathbf{x}_S \right). \tag{4.13}$$

By substituting (4.13) into the first equation of (4.12), we get

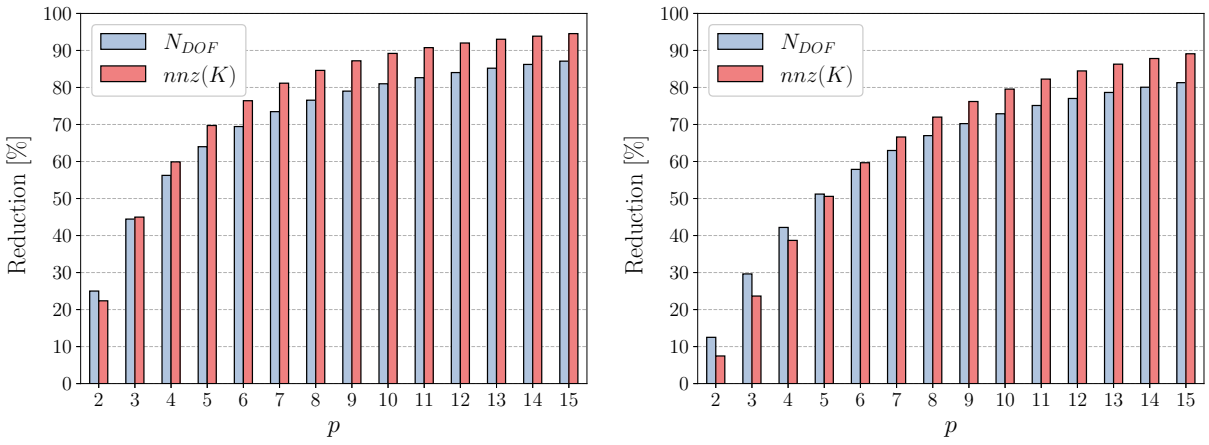
$$\left(\mathbf{K}_S - \mathbf{K}_U \mathbf{K}_D^{-1} \mathbf{K}_L \right) \mathbf{x}_S = \mathbf{f}_S - \mathbf{K}_U \mathbf{K}_D^{-1} \mathbf{f}_D. \tag{4.14}$$

The matrix $\mathbf{K}_S - \mathbf{K}_U \mathbf{K}_D^{-1} \mathbf{K}_L$ is called the Schur complement of the block $\mathbf{K}_D$ of matrix $\mathbf{K}$, and is denoted by $\mathbf{K}/\mathbf{K}_D$. The equation (4.14) allows us to solve for $\mathbf{x}_S$ using the Schur complement as the coefficient matrix, which is smaller than the full matrix $\mathbf{K}$. The remaining unknowns $\mathbf{x}_D$ of the full system can then be computed using (4.13). However, this observation is not very useful on its own, since such a procedure involves inverting the block $\mathbf{K}_D$. The key idea behind static condensation revolves around setting up the decomposition (4.12) in such a way that this inversion is inexpensive to compute. Returning to the finite element context, this can be achieved by setting the row and column indices of $\mathbf{K}_D$ to the set of DOFs associated with internal element nodes, meaning nodes belonging to only one element. Given the index pair (i, j), such that at least one index belongs to this set, the global matrix $\hat{\mathbf{K}}$ will have non-zero entries at (i, j) and (j, i) if and only if there exists an element which contributes to them both. Since there is at most one such element, the matrices $\mathbf{K}_D$, $\mathbf{K}_U$, and $\mathbf{K}_L$ will exhibit a block structure, with each block corresponding to a single element ($\mathbf{K}_D$ will be block-diagonal). This means that the calculation of the Schur complement, and, later, the recovery of the internal DOFs using (4.13), can be performed strictly at the element level, independent of the rest of the mesh. The inverted blocks of $\mathbf{K}_D$ can be stored in memory and retrieved during the latter step.

It should be appreciated that for higher-order expansions, static condensation results not just in the decrease of the size of the algebraic problem, but in the reduction of its order of complexity with regard to p. The number of boundary nodes for quadrilateral and hexahedral elements is equal to $(p+1)^{N_d} - (p-1)^{N_d}$. Since the number of nodes in

the entire mesh is bounded by the sum of the number of nodes of each element, this means that employing static condensation lowers the order of the number of global DOFs from $\mathcal{O}\left(p^{N_d}\right)$ to $\mathcal{O}\left(p^{N_d-1}\right)$. For the same reason, the order of the number of non-zero entries of $\mathbf{K}/\mathbf{K}_D$ is reduced from $\mathcal{O}\left(p^{2N_d}\right)$ to $\mathcal{O}\left(p^{2(N_d-1)}\right)$, compared to the original matrix $\mathbf{K}$. The latter is even more important, since the computational cost of evaluating the action of a sparse matrix is directly proportional to the number of its non-zero entries. The exact reduction in the aforementioned metrics for selected values of p has been quantified in Figure 4.2.

It is also worth emphasizing that static condensation preserves the desirable properties of the algebraic system, since the Schur complement, representing the new coefficient matrix, is also symmetric positive-definite. The symmetry is evident, since $\mathbf{K}_L = \mathbf{K}_U^T$, and $\mathbf{K}_D$ and $\mathbf{K}_S$ are symmetric, being diagonal blocks of the symmetric matrix $\mathbf{K}$. The proof that all eigenvalues of $\mathbf{K}/\mathbf{K}_D$ are positive (and therefore the matrix is positive-definite) can be found in [109].



(a) Quadrilateral elements **(b)** Hexahedral elements

Figure 4.2. Reduction in the number of degrees of freedom N_{DOF} and non-zero stiffness matrix entries $nnz(K)$ resulting from applying static condensation to interior element nodes for varying values of p. Results for Cartesian meshes of (a) quadrilateral and (b) hexahedral elements. Values are reported in the infinite grid size limit.

4.2 The Local Element Approach

The local element approach takes advantage of the fact that the matrix $\hat{\mathbf{K}}$ need not be explicitly formed. Instead, given some vector $\hat{\mathbf{x}} \in \mathbb{R}^{N_g N_u}$, the vector $\hat{\mathbf{y}} \in \mathbb{R}^{N_g N_u}$ is sought, given by the matrix-vector product

$$\hat{\mathbf{y}} = \hat{\mathbf{K}}\hat{\mathbf{x}}. \tag{4.15}$$

The class of methods operating in this fashion are usually referred to as matrix-free methods. By making use of the elementary algebraic property of distributivity of multiplication over addition, the global matrix-vector product can be replaced by a series of local products involving the operators $\mathbf{K}_e$, applied to the relevant entries of $\hat{\mathbf{x}}$. The results of these local products can then be summed into the matching entries of $\hat{\mathbf{y}}$, somewhat analogously to the process of global assembly (with the distinction that the objects being assembled are vectors, not matrices). In each element, the following expression must be evaluated:

$$\mathbf{y}_e = \mathbf{K}_e \mathbf{x}_e, \tag{4.16}$$

where $\mathbf{x}_e$, $\mathbf{y}_e \in \mathbb{R}^{N_l}$. It should be noted that $\mathbf{K}_e$ itself need not be formed explicitly (per (4.10a) and (4.11a), this would require matrix-matrix multiplication). Instead, $\mathbf{y}_e$ can be computed as a sequence of matrix-vector products involving the constituent terms of $\mathbf{K}_e$. For example, the term $\mathbf{y}_e^A \in \mathbb{R}^{N_l}$ corresponding to the domain integral can be written as

$$\mathbf{y}_e^A := \mathbf{M}_A \mathbf{x}_e = \mathbf{H}_A^T \mathbf{W}_A \mathbf{H}_A \mathbf{x}_e. \tag{4.17}$$

The vector $\mathbf{x}_e$ is formed with the help of an appropriate mapping. In [54, p. 188] and related works such as [107], this mapping, denoted by $\mathcal{A}$, is defined at the global level. It is applied to $\hat{\mathbf{x}}$ and produces the vector $\underline{\hat{\mathbf{x}}}^e \in \mathbb{R}^{|\mathcal{E}| N_b N_u}$, composed of blocks corresponding to the elements, each block containing the entries of $\hat{\mathbf{x}}$ associated with the element's DOFs (i.e. the blocks are the vectors $\mathbf{x}_e$). The entries of $\hat{\mathbf{x}}$ will be repeated in $\underline{\hat{\mathbf{x}}}^e$ if the corresponding nodes belong to multiple elements. The element-level operators are applied to the matching blocks, producing $\underline{\hat{\mathbf{y}}}^e \in \mathbb{R}^{|\mathcal{E}| N_b N_u}$, which in turn yields $\hat{\mathbf{y}}$ under the action of the mapping $\mathcal{A}^T$. This makes for elegant mathematical notation, however, it involves the use of the additional vectors $\underline{\hat{\mathbf{x}}}^e$ and $\underline{\hat{\mathbf{y}}}^e$. In the present work, instead of $\mathcal{A}$ and $\mathcal{A}^T$, we employ the element-level mappings $\mathcal{A}_e$ and $\mathcal{A}_e^T$. These are defined so that $\mathbf{x}_e = \mathcal{A}_e \hat{\mathbf{x}}$. Consequently, $\mathcal{A}_e^T \mathbf{y}_e$ is a very sparse vector. The sum of these terms over all elements

yields $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} = \sum_{e \in \mathcal{E}} \mathcal{A}_e^T \mathbf{y}_e. \tag{4.18}$$

This approach allows the gather and scatter operations involved to be performed directly on the vectors $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, eliminating the need for intermediate structures, as depicted in Figure 4.3.

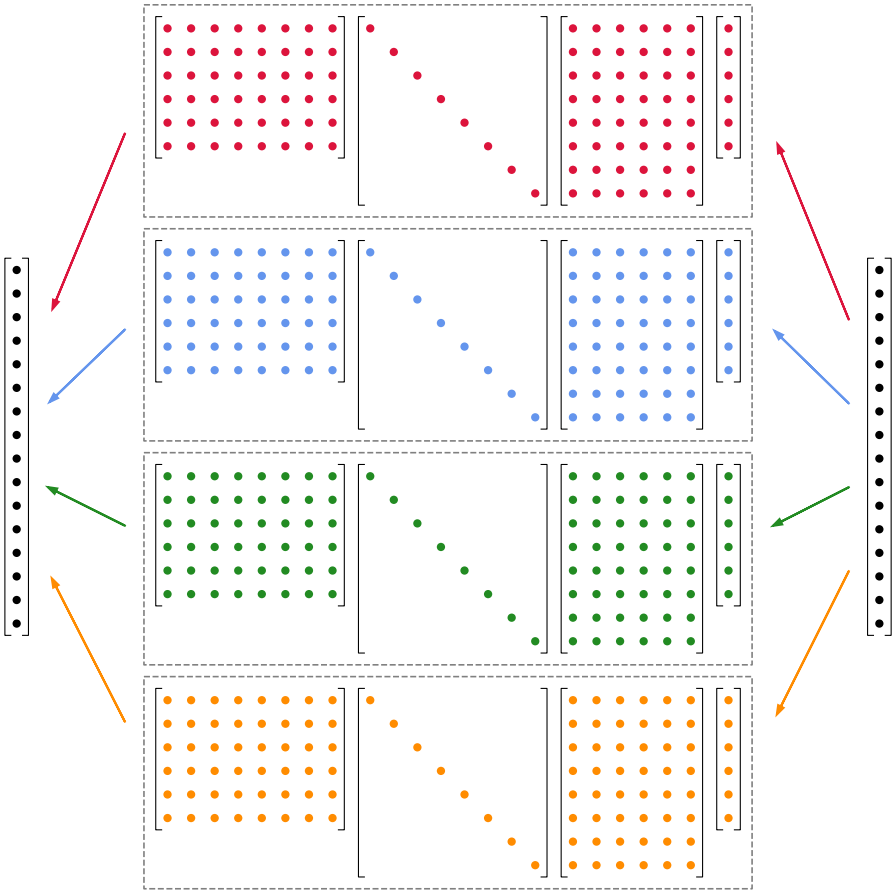


Figure 4.3. Graphical representation of the local element approach. Each color corresponds to a different element. The mappings $\mathcal{A}_e$ and $\mathcal{A}_e^T$ are represented by the arrows on the right and left, respectively.

Two notable benefits of employing the local element approach over global assembly are readily apparent. First, since the matrix $\hat{\mathbf{K}}$ does not need to be stored, the amount of memory needed is dramatically reduced. Second, the number of operations required to evaluate the matrix-free operator is significantly lower than the computational cost of assembling $\hat{\mathbf{K}}$. It would therefore be reasonable to assume that, as long as the number of iterations of the algebraic solver does not exceed some threshold, employing the local

element approach lowers the total solution time. However, this surface-level observation fails to appreciate the efficiency of matrix-free methods to its full extent. Despite requiring strictly fewer arithmetic operations, evaluating an assembled operator involves reading all of $\hat{\mathbf{K}}$ from memory, and is therefore highly memory-bound. For certain types of PDEs and higher values of p, the local element approach may outperform global assembly not only over the course of a limited number of CGM iterations, but even in evaluating a single matrix-vector product, rendering the discussion of an iteration threshold moot. This issue is investigated thoroughly in chapter 5.

4.3 The Sum-Factorization Technique

The sum-factorization technique also belongs to the class of matrix-free operator evaluation strategies. The difference, as compared to the local element approach, lies in the way in which the local operator is applied. Sum-factorization aims to exploit the tensor-product structure of the basis and quadrature to reduce the computational complexity of this operation. The method was first introduced by Orszag [74], and has subsequently been extensively used in the spectral/*hp* element method community. It is of paramount importance to the performance of high-order solvers.

4.3.1 The Underlying Mechanism

To illustrate the key idea behind the sum-factorization technique, we first revisit the reasoning from [20, chapter 4]. Let $\mathbf{A} \in \mathbb{R}^{k\times l}$, $\mathbf{B} \in \mathbb{R}^{m\times n}$ be two real-valued matrices. The tensor (Kronecker) product of $\mathbf{A}$ and $\mathbf{B}$, denoted as $\mathbf{A}\otimes\mathbf{B}$, is the matrix $\mathbf{D} \in \mathbb{R}^{km\times ln}$, such that

$$\mathbf{D} = \mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1l}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \dots & a_{2l}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1}\mathbf{B} & a_{k1}\mathbf{B} & \dots & a_{kl}\mathbf{B} \end{bmatrix}. \tag{4.19}$$

Now let $\mathbf{x} \in \mathbb{R}^{ln}$, and $\mathbf{y} \in \mathbb{R}^{km}$ be the matrix-vector product $\mathbf{y} = \mathbf{D}\mathbf{x}$. If we decompose $\mathbf{x}$ into l blocks of length n, and $\mathbf{y}$ into k blocks of length m, so that

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_l \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_k \end{bmatrix}, \tag{4.20}$$

we can use Equation 4.19 to express this product as

$$\mathbf{y} = (\mathbf{A} \otimes \mathbf{B})\mathbf{x} = \begin{bmatrix} \sum_i^l a_{1i}\mathbf{B}\mathbf{x}_i \\ \sum_i^l a_{2i}\mathbf{B}\mathbf{x}_i \\ \vdots \\ \sum_i^l a_{ki}\mathbf{B}\mathbf{x}_i \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_k \end{bmatrix}. \tag{4.21}$$

If we now reorganize $\mathbf{x}$ and $\mathbf{y}$ into the matrices $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$, respectively, such that

$$\hat{\mathbf{X}} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_l \end{bmatrix}, \quad \hat{\mathbf{Y}} = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \dots & \mathbf{y}_k \end{bmatrix}, \tag{4.22}$$

it follows immediately from Equation 4.21 that

$$\hat{\mathbf{Y}} = \mathbf{B}\hat{\mathbf{X}}\mathbf{A}^T. \tag{4.23}$$

The full matrix-vector product can therefore be computed using 2 matrix-matrix products involving the components of $\mathbf{D}$. After some rearranging, (4.23) leads to

$$\mathbf{U} = \hat{\mathbf{X}}^T\mathbf{B}^T \tag{4.24a}$$

$$\hat{\mathbf{Y}} = \mathbf{U}^T\mathbf{A}^T, \tag{4.24b}$$

where $\mathbf{U} \in \mathbb{R}^{n \times k}$ is an auxiliary matrix containing the intermediate result.

The computational cost of evaluating (4.24) is $2kn(l+m)$ operations (multiplications and additions). By contrast, using a generic approach to calculate $\mathbf{y} = \mathbf{D}\mathbf{x}$ would require $2klmn$ operations. If all matrix dimensions are proportional to some value s, this implies a reduction in complexity from $\mathcal{O}(s^4)$ to $\mathcal{O}(s^3)$.

The sum-factorization approach can also be used to compute the matrix-matrix product $\mathbf{Y} = \mathbf{D}\mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{ln \times r}$, $\mathbf{Y} \in \mathbb{R}^{km \times r}$. Similarly to (4.20), we introduce a block

decomposition of $\mathbf{X}$ into the vectors $\mathbf{x}_{ij}$ of length n:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_{11} & \mathbf{x}_{12} & \dots & \mathbf{x}_{1r} \\ \mathbf{x}_{21} & \mathbf{x}_{22} & \dots & \mathbf{x}_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{l1} & \mathbf{x}_{l2} & \dots & \mathbf{x}_{lr} \end{bmatrix}. \tag{4.25}$$

These can further be arranged into the matrices $\hat{\mathbf{X}}_i \in \mathbb{R}^{n \times l}$, such that

$$\hat{\mathbf{X}}_i = \begin{bmatrix} \mathbf{x}_{1i} & \mathbf{x}_{2i} & \dots & \mathbf{x}_{li} \end{bmatrix}, \quad i \in \{1, 2, \dots, r\}, \tag{4.26}$$

and, finally, into the matrix $\hat{\mathbf{X}} \in \mathbb{R}^{n \times lr}$:

$$\hat{\mathbf{X}} = \begin{bmatrix} \hat{\mathbf{X}}_1 & \hat{\mathbf{X}}_2 & \dots & \hat{\mathbf{X}}_r \end{bmatrix}. \tag{4.27}$$

The simplest way of calculating $\mathbf{Y}$ would be to now apply (4.24) to the individual $\hat{\mathbf{X}}_i$ matrices (in other words, handle the columns of $\mathbf{X}$ separately). However, as will soon become evident, it is useful to formulate an approach which operates on the matrix $\hat{\mathbf{X}}$ as a whole. To this end, following [11], we first introduce the notation $\mathbf{M}^{r_1;c_1}$ to represent the rearrangement of the matrix $\mathbf{M} \in \mathbb{R}^{r_0 \times c_0}$ into an $r_1 \times c_1$ matrix, with the row index running fastest (naturally, this requires that $r_0 c_0 = r_1 c_1$). For example, $\hat{\mathbf{X}} = \mathbf{X}^{n;lr}$. It is worth emphasizing that, assuming all matrices are stored in column-major order, such a reshaping does not require any data movement, merely a change in the stride. Therefore, this operation carries no additional computational cost.

Now, let $\mathbf{U} \in \mathbb{R}^{lr \times m}$ be equal to

$$\mathbf{U} = \hat{\mathbf{X}}^T \mathbf{B}^T = \begin{bmatrix} \hat{\mathbf{X}}_1^T \mathbf{B}^T \\ \hat{\mathbf{X}}_2^T \mathbf{B}^T \\ \vdots \\ \hat{\mathbf{X}}_r^T \mathbf{B}^T \end{bmatrix}. \tag{4.28}$$

The blocks $\hat{\mathbf{X}}_i^T \mathbf{B}^T$ clearly correspond to (4.24a). However, (4.24b) cannot be applied directly, since the block layout and matrix dimensions are incompatible. Instead, we must reshape $\mathbf{U}$ before transposing it and right-multiplying by $\mathbf{A}^T$. Let $\mathbf{u}_{ij}$ denote the

j-th column of $\hat{\mathbf{X}}_i^T \mathbf{B}^T$. It follows from (4.28) that

$$\left(\mathbf{V}^{l;mr}\right)^T \mathbf{A}^T = \begin{bmatrix} \mathbf{u}_{11}^T \mathbf{A}^T \\ \mathbf{u}_{21} \mathbf{A}^T \\ \vdots \\ \mathbf{u}_{r1} \mathbf{A}^T \\ \mathbf{u}_{12} \mathbf{A}^T \\ \mathbf{u}_{22} \mathbf{A}^T \\ \vdots \\ \mathbf{u}_{mr} \mathbf{A}^T \end{bmatrix}. \tag{4.29}$$

This expression contains all the desired entries of $\mathbf{Y}$, albeit in an incorrect arrangement. In order to recover $\mathbf{Y}$, all that is needed is another reshaping and a transposition, leading to the following steps for computing $\mathbf{Y} = (\mathbf{A} \otimes \mathbf{B})\,\mathbf{X}$:

$$\mathbf{U} = \left(\mathbf{X}^{n;lr}\right)^T \mathbf{B}^T, \tag{4.30a}$$

$$\mathbf{V} = \left(\mathbf{U}^{l;mr}\right)^T \mathbf{A}^T, \tag{4.30b}$$

$$\mathbf{Y} = \left(\mathbf{V}^{r;km}\right)^T, \tag{4.30c}$$

where $\mathbf{V} \in \mathbb{R}^{mr \times k}$ is another auxiliary matrix. Note that, as established in (4.24), for $r = 1$ the transposition in the last step can be elided, since $\mathbf{y} = \left(\mathbf{V}^{1;km}\right)^T = \mathbf{V}^{km;1}$.

Finally, let us consider the case of the triple tensor product. Let $\mathbf{C} \in \mathbb{R}^{o \times p}$ and now $\mathbf{D} = \mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C}$, $\mathbf{X} \in \mathbb{R}^{lnp \times r}$, and $\mathbf{Y} \in \mathbb{R}^{kmo \times r}$. Grouping the products as $\mathbf{D} = (\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C}$ and recursively applying (4.30) immediately yields the following steps:

$$\mathbf{U} = \left(\mathbf{X}^{p;lnr}\right)^T \mathbf{C}^T, \tag{4.31a}$$

$$\mathbf{V} = \left(\mathbf{U}^{n;lor}\right)^T \mathbf{B}^T, \tag{4.31b}$$

$$\mathbf{W} = \left(\mathbf{V}^{l;mor}\right)^T \mathbf{A}^T, \tag{4.31c}$$

$$\mathbf{Y} = \left(\mathbf{W}^{r;kmo}\right)^T, \tag{4.31d}$$

where $\mathbf{W} \in \mathbb{R}^{mor \times k}$ is yet another auxiliary matrix. Evaluating (4.31) requires $2or(lnp + lmn + klm)$ arithmetic operations. If all dimensions of the constituent matrices of $\mathbf{D}$ are proportional to some value s, this results in a computational complexity of $\mathcal{O}(rs^4)$, compared to $\mathcal{O}(rs^6)$ when using the generic approach.

4.3.2 Application to the LSFEM operator

Having established how to leverage the tensor-product structure of a matrix when computing its product with a vector, let us now employ this technique to evaluate the least-squares operator in quadrilateral and hexahedral elements. The following discussion is limited to computing to the domain integral term $\mathbf{y}_e^A$, defined in (4.17), which carries the majority of the computational cost. Recall that in one dimension, the bases in reference space were denoted by the functions ψ_i, defined in (3.11). Let the matrices $\boldsymbol{\Psi}, \boldsymbol{\Psi}' \in \mathbb{R}^{p+1\times q}$ tabulate, respectively, the values and derivatives of the one-dimensional basis at the abscissas of the quadrature rule, so that

$$\Psi_{ij} = \psi_i\left(q_j\right), \tag{4.32a}$$

$$\Psi'_{ij} = \frac{\partial \psi_i}{\partial \xi}\left(q_j\right). \tag{4.32b}$$

$\boldsymbol{\Psi}$ and $\boldsymbol{\Psi}'$ are often referred to as the interpolation and derivative matrices, respectively. Now let $\mathbf{R}_i \in \mathbb{R}^{N_b\times N_q}$, $i = 0, \ldots, N_d$ denote the matrices containing the values of the reference bases and their derivatives at the quadrature points of the higher-dimensional elements, meaning that

$$\left(r_0\right)_{ij} = \varphi_i\left(\mathbf{q}_j\right), \tag{4.33a}$$

$$\left(r_k\right)_{ij} = \frac{\partial \varphi_i}{\partial \xi_k}\left(\mathbf{q}_j\right),\ k = 1, \ldots, N_d. \tag{4.33b}$$

Per (3.13) and (3.14), the bases in quadrilateral and hexahedral elements take the forms $\varphi_{ij}\left(\xi, \eta\right) = \psi_i\left(\xi\right)\psi_j\left(\eta\right)$ and $\varphi_{ijk}\left(\xi, \eta, \zeta\right) = \psi_i\left(\xi\right)\psi_j\left(\eta\right)\psi_k\left(\zeta\right)$, respectively. The quadrature points conform to a similar structure, as indicated in (3.34) and (3.35). Given the above, the matrices $\mathbf{R}_i$ for quadrilateral elements can be expressed as

$$\mathbf{R}_0 = \boldsymbol{\Psi} \otimes \boldsymbol{\Psi}, \tag{4.34a}$$

$$\mathbf{R}_1 = \boldsymbol{\Psi} \otimes \boldsymbol{\Psi}', \tag{4.34b}$$

$$\mathbf{R}_2 = \boldsymbol{\Psi}' \otimes \boldsymbol{\Psi}, \tag{4.34c}$$

while for hexahedral elements, they take the form

$$\mathbf{R}_0 = \boldsymbol{\Psi} \otimes \boldsymbol{\Psi} \otimes \boldsymbol{\Psi}, \tag{4.35a}$$

$$\mathbf{R}_1 = \boldsymbol{\Psi} \otimes \boldsymbol{\Psi} \otimes \boldsymbol{\Psi}', \tag{4.35b}$$

$$\mathbf{R}_2 = \boldsymbol{\Psi} \otimes \boldsymbol{\Psi}' \otimes \boldsymbol{\Psi}, \tag{4.35c}$$

$$\mathbf{R}_3 = \boldsymbol{\Psi}' \otimes \boldsymbol{\Psi} \otimes \boldsymbol{\Psi}. \tag{4.35d}$$

It is the tensor products present in (4.34) and (4.35) which we seek to exploit. Recalling (4.5), it should be noted that while $\mathbf{N}_0 = \mathbf{R}_0$, the matrices $\mathbf{N}_i$ and $\mathbf{R}_i$ for $i = 1, \ldots, N_d$ are distinct, as they represent derivatives in physical and reference space, respectively. In particular, the physical derivatives do not exhibit a tensor-product structure. For this reason, and due to the fact that the matrices describing the differential operator may not be constant, the sum-factorization technique cannot be directly applied to the matrix $\mathbf{H}_A$ present in (4.10). Instead, each of its constituent terms $\boldsymbol{\kappa}_i$ must be considered separately. We start by rewriting (4.6) in the following matrix form:

$$\boldsymbol{\kappa}_i = \operatorname{diag}\left(\mathbf{A}_i\left(\mathbf{q}_1\right), \ldots, \mathbf{A}_i\left(\mathbf{q}_{N_q}\right)\right)\left(\mathbf{N}_i^T \otimes \mathbf{I}_{N_u}\right), \tag{4.36}$$

where $\mathbf{I}_{N_u}$ denotes the identity matrix of size N_u. Introducing $\boldsymbol{\iota} = \mathbf{J}_e^{-T}$ to make the subsequent notation less cluttered, (4.5) can similarly be rewritten as

$$\mathbf{N}_0 = \mathbf{R}_0, \tag{4.37a}$$

$$\mathbf{N}_i = \sum_{j=1}^{N_d} \mathbf{R}_j \operatorname{diag}\left(\iota_{ij}\left(\mathbf{q}_1\right), \ldots, \iota_{ij}\left(\mathbf{q}_{N_q}\right)\right), \; i = 1, \ldots, N_d. \tag{4.37b}$$

Next, the following block-diagonal matrices $\mathbf{D}_0, \mathbf{D}_{ij} \in \mathbb{R}^{N_q N_e \times N_q N_u}$, $i, j = 1, \ldots, N_d$ are defined:

$$\mathbf{D}_0 = \operatorname{diag}\left(\mathbf{A}_0\left(\mathbf{q}_1\right), \ldots, \mathbf{A}_0\left(\mathbf{q}_{N_q}\right)\right), \tag{4.38a}$$

$$\mathbf{D}_{ij} = \operatorname{diag}\left(\mathbf{A}_i\left(\mathbf{q}_1\right)\iota_{ij}\left(\mathbf{q}_1\right), \ldots, \mathbf{A}_i\left(\mathbf{q}_{N_q}\right)\iota_{ij}\left(\mathbf{q}_{N_q}\right)\right). \tag{4.38b}$$

Substituting (4.37) into (4.36), and observing that the diagonal terms can be combined, results in

$$\boldsymbol{\kappa}_0 = \mathbf{D}_0\left(\mathbf{R}_0^T \otimes \mathbf{I}_{N_u}\right), \tag{4.39a}$$

$$\boldsymbol{\kappa}_i = \sum_{j=1}^{N_d} \mathbf{D}_{ij}\left(\mathbf{R}_j^T \otimes \mathbf{I}_{N_u}\right), \; i = 1, \ldots, N_d. \tag{4.39b}$$

After factoring out the terms $\left(\mathbf{R}_j^T \otimes \mathbf{I}_{N_u}\right)$, (4.7) now becomes

$$\mathbf{H}_A = \sum_{i=0}^{N_d} \boldsymbol{\kappa}_i = \mathbf{D}_0\left(\mathbf{R}_0^T \otimes \mathbf{I}_{N_u}\right) + \sum_{j=1}^{N_d}\left(\sum_{i=1}^{N_d} \mathbf{D}_{ij}\right)\left(\mathbf{R}_j^T \otimes \mathbf{I}_{N_u}\right). \tag{4.40}$$

Consequently, $\mathbf{y}_e^A = \mathbf{H}_A^T \mathbf{W}_A \mathbf{H}_A \mathbf{x}_e$ can be obtained in the following steps:

1. Compute the vectors $\mathbf{t}_i \in \mathbb{R}^{N_q N_u}$, $i = 0, \ldots, N_d$, given by

$$\mathbf{t}_i = \left(\mathbf{R}_i^T \otimes \mathbf{I}_{N_u}\right)\mathbf{x}_e. \tag{4.41}$$

2. Form the block-diagonal matrices $\hat{\mathbf{D}}_i \in \mathbb{R}^{N_q N_e \times N_q N_u}$, $i = 1, \ldots, N_d$, defined as

$$\hat{\mathbf{D}}_i = \sum_{j=1}^{N_d} \mathbf{D}_{ji},\ i = 1, \ldots, N_d \tag{4.42}$$

3. Calculate the vector $\hat{\mathbf{t}} \in \mathbb{R}^{N_q N_e}$, using

$$\hat{\mathbf{t}} = \mathbf{W}_A \left(\mathbf{D}_0 \mathbf{t}_0 + \sum_{i=1}^{N_d} \hat{\mathbf{D}}_i \mathbf{t}_i \right) \tag{4.43}$$

4. Form the vectors $\tilde{\mathbf{t}}_i \in \mathbb{R}^{N_q N_u}$, $i = 0, \ldots, N_d$, given by

$$\tilde{\mathbf{t}}_0 = \mathbf{D}_0^T \hat{\mathbf{t}}, \tag{4.44a}$$

$$\tilde{\mathbf{t}}_i = \hat{\mathbf{D}}_i^T \hat{\mathbf{t}},\ i = 0, \ldots, N_d. \tag{4.44b}$$

5. Compute the desired result by employing

$$\mathbf{y}_e^A = \sum_{i=0}^{N_d} \left(\mathbf{R}_i \otimes \mathbf{I}_{N_u} \right) \tilde{\mathbf{t}}_i. \tag{4.45}$$

The algorithm outlined above proceeds in three phases, as has been depicted schematically in Figure 4.4. The first step can be interpreted as the evaluation of the vector field implied by $\mathbf{x}_e$, along with its derivatives in reference space, at the quadrature points. It requires $N_d + 1$ products involving the matrices $\mathbf{R}_i$. By leveraging their tensor-product structure and employing (4.30) or (4.31), this step can be evaluated in terms of $N_d \left(N_d + 1 \right)$ smaller matrix-matrix multiplications. Following [59], these products will be referred to as sum-factorization sweeps throughout the remainder of this work. Their total cost is $6N_u qp(p + q)$ and $8N_u q(p^3 + qp^2 + q^2 p)$ algebraic operations for quadrilateral and hexahedral elements, respectively. A further examination of the structure of (4.34) and (4.35) reveals that some of the sum-factorization sweeps are redundant, yielding respective reductions in cost by factors of approximately $\frac{1}{6}$ and $\frac{1}{4}$. The second phase, consisting of steps 2 through 4, invokes the coordinate mapping and partial differential operators at the abscissas. Critically, the matrices involved are all block-diagonal, meaning that their action at quadrature points is entirely decoupled. Since the number of operations at a given abscissa does not depend on the element order, the overall computational complexity of the second phase is $\mathcal{O}\left(q^{N_d}\right)$. The final phase consists of the evaluation of the transpose of the operators involved in the first step, and consequently requires $6N_u qp(p + q)$ and

$8N_u p(q^3 + pq^2 + p^2 q)$ algebraic operation for quadrilateral and hexahedral elements, respectively. Since the results are added together at the end, analogous redundancies as before can be leveraged, yielding the same savings in cost.

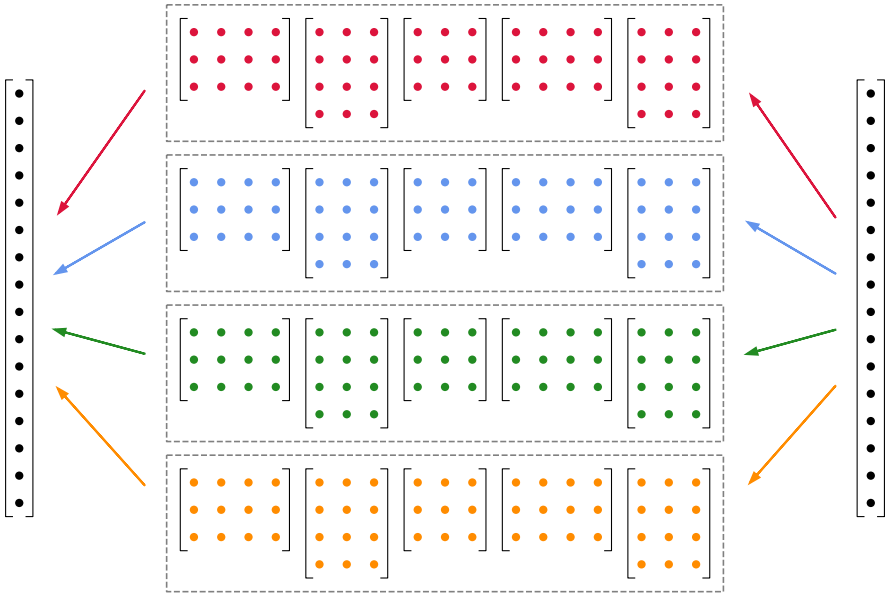


Figure 4.4. Graphical representation of the sum-factorization technique. Matrix-vector products present in the local element approach are replaced by matrix-matrix operations. The role of the mappings $\mathcal{A}_e$ and $\mathcal{A}_e^T$ remains unchanged.

4.3.3 Basis Evaluation Using an Odd-Even Decomposition

Sum-factorization sweeps, which contribute the bulk of the cost of the algorithm described above, consist of right-multiplication by $\mathbf{\Psi}$ and $\mathbf{\Psi}'$ and their transposes. Owing to the way these matrices were constructed, they exhibit certain symmetries that can be exploited to reduce the computational cost of applying them by approximately half. This idea was first proposed in [97], and a robust discussion of its application can be found in [58, chapter 3]. The key observation enabling this technique is that both the basis nodes and quadrature points are symmetrically distributed about the origin. Consequently, the one-dimensional bases are pairwise reflections about the y-axis. Emphasizing that 1-based indexing is used throughout this work, this relationship can be expressed as

$$\forall_{x\in[-1,1]}\, \psi_i(x) = \psi_{p+2-i}(-x). \tag{4.46}$$

The following properties of the interpolation and derivative matrices follow from (4.46) and the symmetric distribution of the abscissas:

$$\Psi_{ij} = \Psi_{p+2-i,q+1-j}, \tag{4.47a}$$

$$\Psi'_{ij} = -\Psi'_{p+2-i,q+1-j}. \tag{4.47b}$$

Let us now show how to efficiently compute the product $\mathbf{y} = \mathbf{x}^T\boldsymbol{\Psi}$, for some vector $\mathbf{x} \in \mathbb{R}^{p+1}$. The generalization to the matrix-matrix product is entirely straightforward, this simplification is merely intended to make the notation more clear. It should also be noted that the following approach can be used without change to evaluate the action of $\boldsymbol{\Psi}^T$, since the property (4.47a) holds under transposition. We start with the case where the number of bases $p+1$ is even, and recall the reasoning from the original paper [97]. Let $\mathbf{o}, \mathbf{e} \in \mathbb{R}^{p+1}$ be the odd and even parts of $\mathbf{x}$, respectively, so that

$$o_i = \frac{1}{2}\left(x_i - x_{p+2-i}\right), \tag{4.48a}$$

$$e_i = \frac{1}{2}\left(x_i + x_{p+2-i}\right). \tag{4.48b}$$

This decomposition is constructed so that

$$o_i = -o_{p+2-i}, \tag{4.49a}$$

$$e_i = e_{p+2-i}. \tag{4.49b}$$

Now let $\mathbf{o}', \mathbf{e}' \in \mathbb{R}^q$ be the row vectors

$$\mathbf{o}' = \mathbf{o}^T\boldsymbol{\Psi}, \tag{4.50a}$$

$$\mathbf{e}' = \mathbf{e}^T\boldsymbol{\Psi}. \tag{4.50b}$$

Using (4.47a) and (4.49), $\mathbf{o}'$ and $\mathbf{e}'$ can be shown to share the odd/even property of $\mathbf{o}$ and

$\mathbf{e}$, respectively:

$$o'_i = \sum_{j=1}^{p+1} \Psi_{ji} o_j = \sum_{j=1}^{\frac{p+1}{2}} \Psi_{ji} o_j + \Psi_{p+2-j,i} o_{p+2-j} = \sum_{j=1}^{\frac{p+1}{2}} \underbrace{\left(\Psi_{ji} - \Psi_{p+2-j,i}\right)}_{\mathbf{\Psi}^-} o_j$$

$$= -\left(\sum_{j=1}^{\frac{p+1}{2}} \Psi_{p+2-j,q+1-i} o_{p+2-j} + \Psi_{j,q+1-i} o_j\right) = -\sum_{j=1}^{p+1} \Psi_{j,q+1-i} o_j = -o'_{q+1-i}, \quad (4.51a)$$

$$e'_i = \sum_{j=1}^{p+1} \Psi_{ji} e_j = \sum_{j=1}^{\frac{p+1}{2}} \Psi_{ji} e_j + \Psi_{p+2-j,i} e_{p+2-j} = \sum_{j=1}^{\frac{p+1}{2}} \underbrace{\left(\Psi_{ji} + \Psi_{p+2-j,i}\right)}_{\mathbf{\Psi}^+} e_j$$

$$= \sum_{j=1}^{\frac{p+1}{2}} \Psi_{p+2-j,q+1-i} e_{p+2-j} + \Psi_{j,q+1-i} e_j = \sum_{j=1}^{p+1} \Psi_{j,q+1-i} e_j = e'_{q+1-i}. \quad (4.51b)$$

If q is even, this means that exactly half ($\frac{q}{2}$) of these vectors' entries need to be calculated. In the event that q is odd, the middle (i.e., $\left\lceil \frac{q}{2} \right\rceil$-th) entry of $\mathbf{o}'$ is zero, since it is equal to its negation. Consequently, $\left\lfloor \frac{q}{2} \right\rfloor$ and $\left\lceil \frac{q}{2} \right\rceil$ entries of $\mathbf{o}'$ and $\mathbf{e}'$, respectively, must be computed. In both cases, each of the halves of $\mathbf{o}'$ and $\mathbf{e}'$ can be computed by right-multiplying the (transposed) halves of the vectors $\mathbf{o}$ and $\mathbf{e}$ by the matrices $\mathbf{\Psi}^- \in \mathbb{R}^{\frac{p+1}{2} \times \left\lfloor \frac{q}{2} \right\rfloor}$ and $\mathbf{\Psi}^+ \in \mathbb{R}^{\frac{p+1}{2} \times \left\lceil \frac{q}{2} \right\rceil}$, respectively, as indicated in (4.51). The total cost incurred by these operations is roughly half that of directly computing $\mathbf{x}^T \mathbf{\Psi}$. Since $\mathbf{x} = \mathbf{o} + \mathbf{e}$, it follows from linearity that $\mathbf{y} = \mathbf{o}' + \mathbf{e}'$. $\mathbf{y}$ can therefore be reconstructed from

$$y_i = o'_i + e'_i, \quad (4.52a)$$

$$y_{q+1-i} = e'_i - o'_i, \quad (4.52b)$$

for $i = 1, \ldots, \left\lceil \frac{q}{2} \right\rceil$.

So far, $p+1$ has been assumed to be even. In the event that it is odd, the procedure outlined above must be modified, since the sums present in (4.51) can no longer be split into two parts of equal size. If we let these sums run until $\left\lceil \frac{p+1}{2} \right\rceil$, the final term will be counted twice. In the case of (4.51a), this issue resolves itself, since the $\left\lceil \frac{p+1}{2} \right\rceil$-th entry of $\mathbf{o}$ is zero. $\mathbf{\Psi}^-$ therefore becomes a $\left\lfloor \frac{p+1}{2} \right\rfloor$ by $\left\lfloor \frac{q}{2} \right\rfloor$ matrix. In (4.51b), the last term must be divided by 2, meaning that $\mathbf{\Psi}^+$ is a $\left\lceil \frac{p+1}{2} \right\rceil$ by $\left\lceil \frac{q}{2} \right\rceil$ matrix whose last row is $\Psi_{\left\lceil \frac{p+1}{2} \right\rceil, i}$.

The same reasoning can be applied to evaluating right-multiplication by $\mathbf{\Psi}'$. The key difference is that the odd/even property of the vectors $\mathbf{o}'$ and $\mathbf{e}'$ is reversed (i.e., $\mathbf{o}'$ is even and $\mathbf{e}'$ is odd), due to the symmetry of $\mathbf{\Psi}'$ being governed by (4.47b), not (4.47a). Consequently, $\mathbf{\Psi}^-$ becomes a $\left\lfloor \frac{p+1}{2} \right\rfloor$ by $\left\lceil \frac{q}{2} \right\rceil$ matrix, $\mathbf{\Psi}^+$ becomes a $\left\lceil \frac{p+1}{2} \right\rceil$ by $\left\lfloor \frac{q}{2} \right\rfloor$ matrix, and the sign of (4.52b) must be negated.

4.4 Comparison of Computational Cost

To conclude this chapter, let us compare the computational costs of employing the different strategies to evaluate the action of the discrete operators arising from the application of least-squares formulation to the various governing equations described in chapter 2. Two metrics will be analyzed for differing approximation orders: the number of required floating-point operations (flops[1]) and the memory bandwidth needed to sustain the computation. Together, these provide an *a priori* estimate of the efficiency of a given approach [56]. In addition to the value of p, several characteristics of the PDEs under consideration influence the aforementioned indicators. In the case of assembled approaches, where the global stiffness matrices are computed ahead of time, only the number of unknowns N_u affects their structure. However, when evaluating the operator in a matrix-free fashion, the number of equations N_e and number of quadrature abscissas, governed by q (recalling that $N_q = q^{N_d}$), are also relevant. In the present work, the value of q is chosen so that the Gauss-Legendre quadrature is exact. The order of the integrand follows directly from (3.5). Furthermore, some operators may require the values and/or derivatives of known external fields. For example, when solving the advection-diffusion problem, the values of the advection velocity at the quadrature points must be known. If these fields are not prescribed in a closed form by functions of space and time, but are instead results of prior solves described using the approximation, then their approximating weights must be read from memory, and their values and/or derivatives computed. The number of such scalar fields is denoted by N_f. It is important to note that N_f accounts only for the fields required by the operator, not those associated with the right-hand side; hence, it remains independent of the time discretization scheme's order. The values of the parameters discussed above for the different governing equations have been shown in Table 4.2.

The number of flops required to evaluate a sparse matrix-vector product is equal to twice the number of the matrix's non-zero entries (for every entry, one multiplication with the corresponding element of the vector and one summation into an accumulator). The matrices arising in the assembled approaches are composed of N_u by N_u blocks, with their adjacency graph determined by the structure of the mesh. For a fixed mesh, the number of nodes is proportional to p^{N_d}. In terms of blocks, the number of non-zero entries in the

[1]The abbreviation "flop" is used to describe a single floating-point operation, and its plural should not be confused with flop/s, meaning floating-points operations *per second*.

Equation	N_u	N_e	N_f	q
2D diffusion	3	4	0	p
2D advection-diffusion	3	4	2	$2p$
2D div-curl	2	3	0	$p+1$
2D Navier-Stokes	4	4	2	$2p$
3D diffusion	4	7	0	p
3D advection-diffusion	4	7	3	$2p$
3D div-curl	3	4	0	$p+1$
3D Navier-Stokes	7	8	3	$2p$

Table 4.2. Comparison of different equations: number of unknowns, equations, external fields, and Gauss-Legendre quadrature points in one dimension required for exact integration.

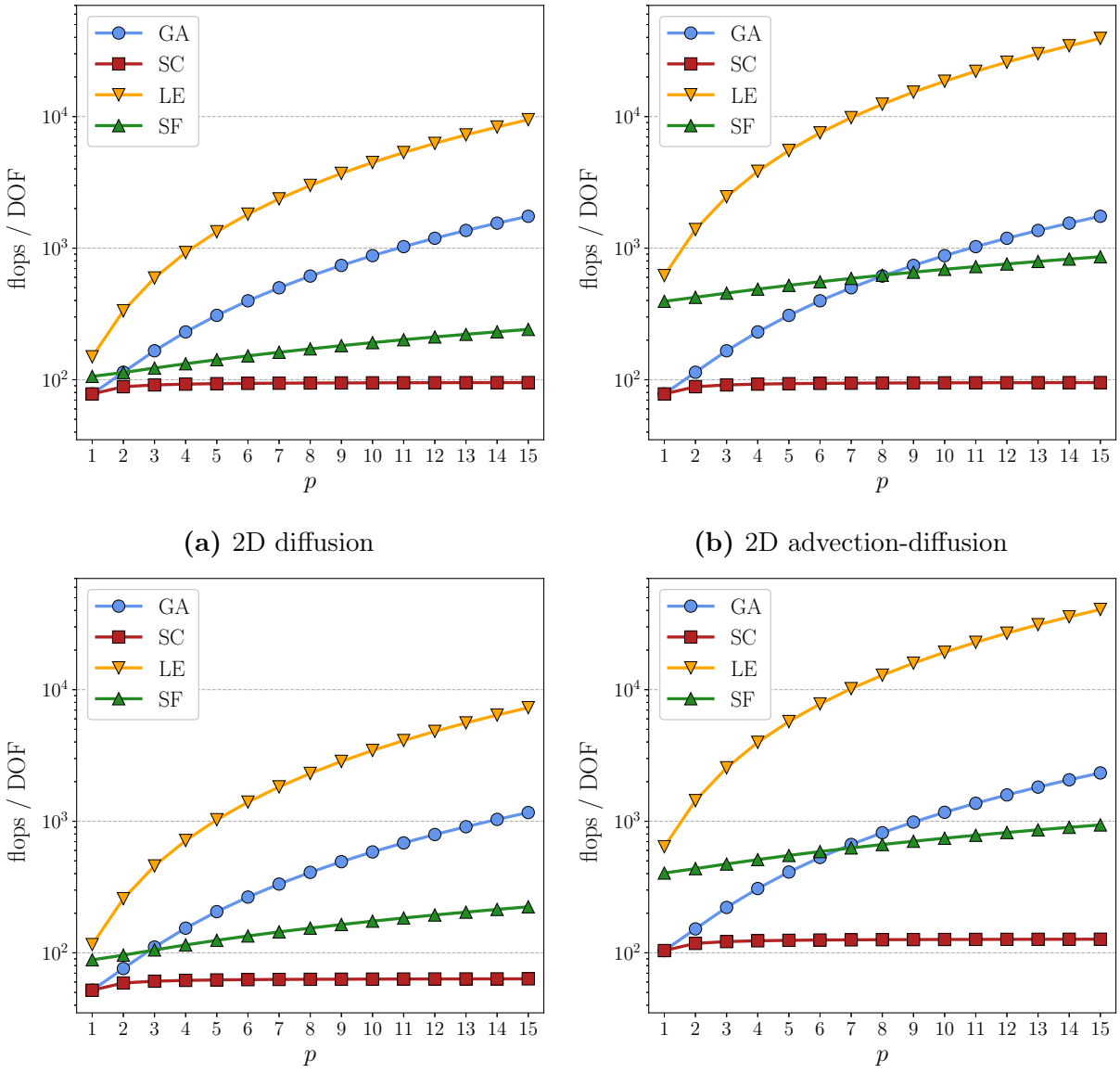
i-th row is equal to the number of nodes which share an element with node i (including itself). In well-defined meshes, the number of elements which a node can belong to is bounded by a small constant (e.g. 2^{N_d} for Cartesian meshes), meaning that the total number of non-zero blocks is proportional to p^{2N_d}. The computational complexity of the global assembly approach is therefore $\mathcal{O}\left(p^{2N_d}N_u^2\right)$ flops. As discussed in subsection 4.1.1, employing static condensation brings this figure down to $\mathcal{O}\left(p^{2(N_d-1)}N_u^2\right)$. The amount of memory which is required to store the matrix's non-zero entries – and later retrieve them for computation – depends on the storage scheme. In the present work, we employ the popular compressed sparse row (CSR) format [96]. The non-zero entries are stored in row-major order along with the index of their column. The rows are delimited with the help of an additional array whose size is equal to the number of rows plus 1. This value is much smaller than the number of entries, especially for higher p, and can therefore be neglected in the analysis. It is worth noting that, to accommodate multi-physics problems in which nodes may have varying numbers of DOFs, the CSR format is employed at the individual entry level instead of the block level. Assuming the use of double precision for the matrix entries and 32-bit integers for the column indices, the approach outlined above requires 12 bytes per matrix entry. The arithmetic intensity is therefore fixed at 1 operation per 6 bytes, independent of p.

In the local element approach, the values and derivatives of the external fields must first be computed, which involves $2N_f(N_d+1)(p+1)^{N_d}q^{N_d}$ flops. Next, the matrix $\mathbf{H}_A$ is formed at the cost of $(2N_d+1)\,N_uN_e(p+1)^{N_d}q^{N_d}$ multiplications and additions. Subsequently, the expression (4.17) must be evaluated, increasing the cost by a further $\left(4N_u(p+1)^{N_d}+2\right)q^{N_d}N_e$ operations. Taking into account that $q \propto p$, the computational complexity of the local element approach is $\mathcal{O}\left(p^{2N_d}N_uN_e\right)$. Employing the sum-factorization technique reduces this figure. The computation of the external fields can be performed as part of the sum-factorization sweeps constituting the first step of the procedure outlined in subsection 4.3.2. Their approximation weights are placed alongside the entries of $\mathbf{x}_e$ corresponding to the relevant nodes, effectively increasing the number of DOFs per node by N_f during the first phase. The overall computational complexity therefore remains unaffected at $\mathcal{O}\left(p^{N_d+1}N_u\right)$. Both matrix-free strategies require the same data describing an element in order to evaluate the action of its local operator. First, the indices of the element's nodes, stored as 32-bit integers, must be retrieved. By employing a scheme wherein the internal nodes are assigned consecutive identifiers, the number of required memory accesses can be limited to $(p+1)^{N_d-1}+1$. Next, the corresponding entries of $\hat{\mathbf{x}}$ needed to form $\mathbf{x}_e$, along with the approximation weights of the external fields must be read. This involves reading $(p+1)^{N_d}(N_u+N_f)$ values from memory. Finally, the geometric information needed to compute the Jacobian matrix must be retrieved. In the present work, given that the scope is limited to linear elements, only the geometric approximation weights (vertex coordinates) $\mathbf{V}_e$ are stored, and $\mathbf{J}_e$, along with its inverse and determinant, is computed on the fly. A discussion of alternative approaches, where $\mathbf{J}_e$ is precomputed at every quadrature point can be found in [28]. Once $\mathbf{y}_e$ is computed, it must be scattered into $\hat{\mathbf{y}}$, which requires a further $(p+1)^{N_d}N_u$ accesses. All other data structures, which are shared between elements, and whose memory access is therefore amortized across the loop over the mesh, are discounted in the memory bandwidth analysis[2]. Consequently, the memory complexity of matrix-free evaluation strategies is $\mathcal{O}\left(p^{N_d}N_u\right)$, and their arithmetic intensity increases with the approximation order.

The number of flops per DOF required by the different evaluation strategies, assuming a large Cartesian mesh, has been shown in Figure 4.5 and Figure 4.6 for two-dimensional

[2]Some of these structures, such as the matrices $\mathbf{\Psi}$, are global constants, entirely independent of the element being considered. The entries of others, such as $\mathbf{H}_A$, are computed at every element. However, the allocation used to store them can be reused.

and three-dimensional problems, respectively. It can be seen that for two-dimensional problems, the global assembly approach with static condensation requires the fewest flops for all equations considered. However, in three dimensions, the sum-factorization technique is either more efficient outright (diffusion and div-curl equations), or becomes more efficient at higher orders ($p = 10$ and $p = 6$ for the advection-diffusion and Navier-Stokes equations, respectively). The memory required per DOF has been shown in Figure 4.7 and Figure 4.8. The matrix-free approaches are significantly more memory-efficient, and this behavior becomes even more pronounced as p increases. It can be expected that even for problems where their computational cost is nominally lower, the assembled strategies will exhibit poorer performance due to memory bandwidth limitations. This issue is investigated experimentally in the next chapter.



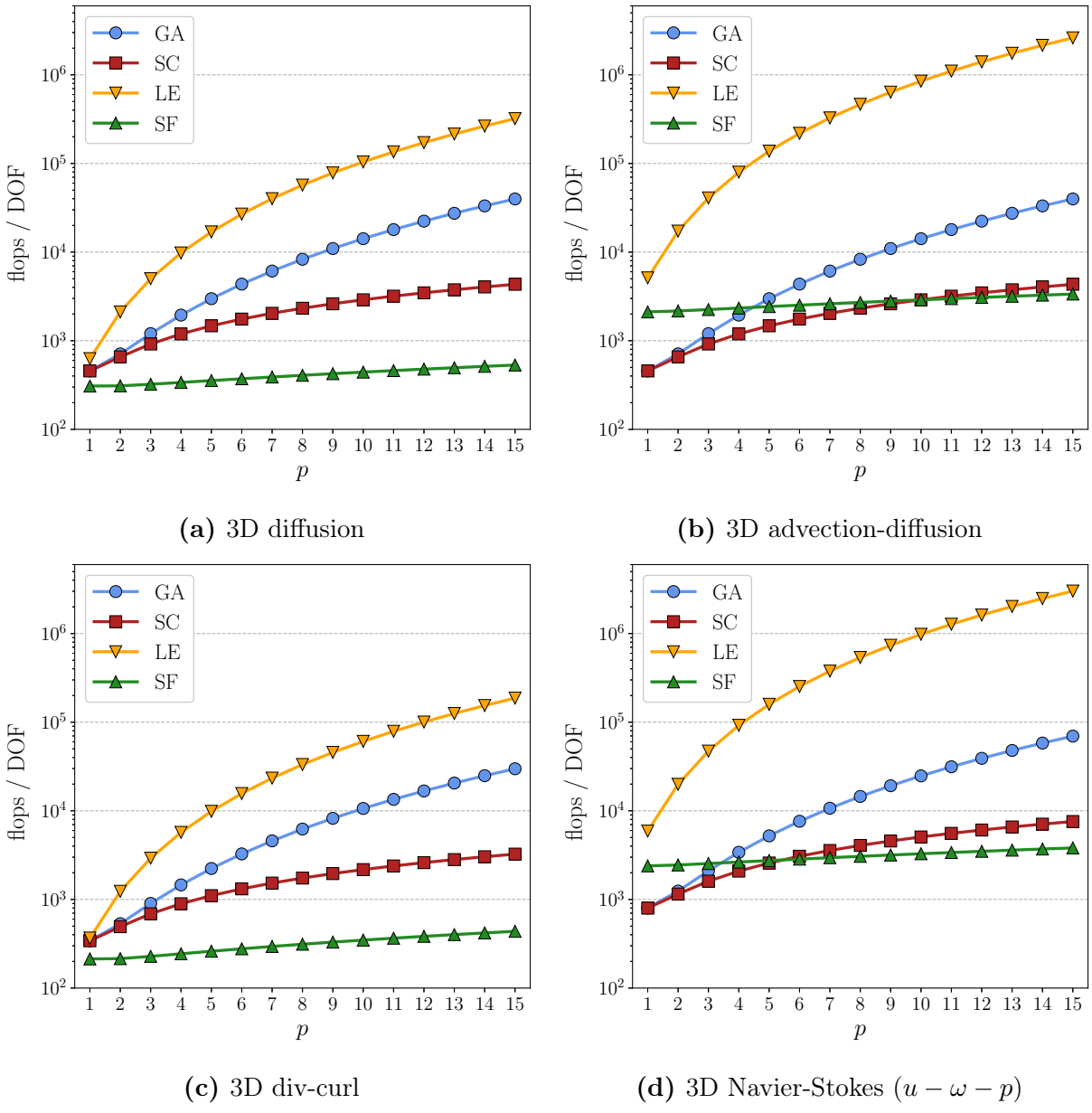
(a) 2D diffusion

(b) 2D advection-diffusion

(c) 2D div-curl

(d) 2D Navier-Stokes ($u - \omega - p$)

Figure 4.5. Comparison of floating-point operations per DOF required to evaluate the least-squares operators associated with different 2D governing equations for varying element orders using the strategies described in this chapter: global assembly (GA), static condensation (SC), local element (LE), and sum-factorization (SF). Results for the sum-factorization technique include the savings from employing the odd-even decomposition.



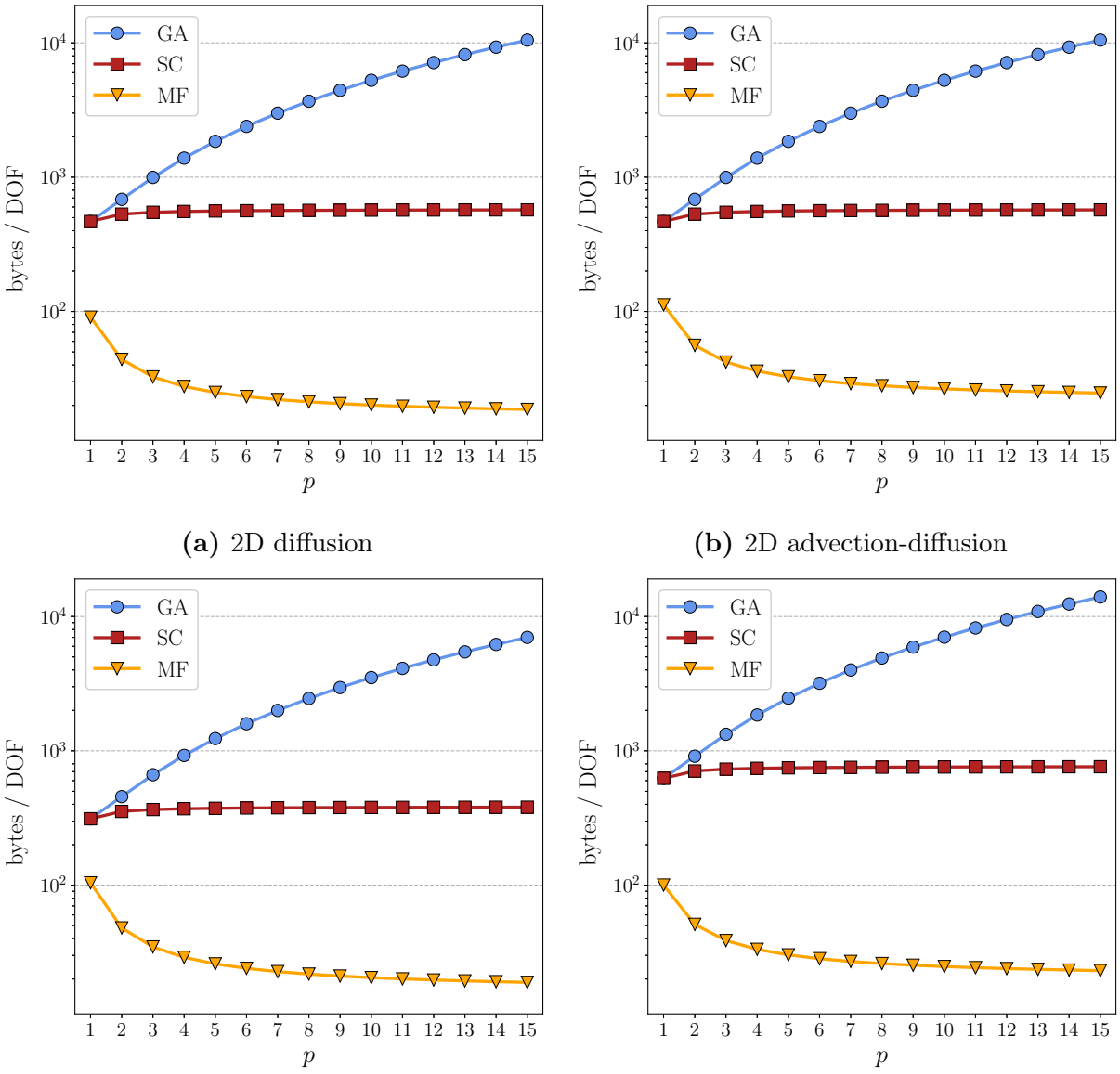
(a) 3D diffusion

(b) 3D advection-diffusion

(c) 3D div-curl

(d) 3D Navier-Stokes ($u - \omega - p$)

Figure 4.6. Comparison of floating-point operations per DOF required to evaluate the least-squares operators associated with different 3D governing equations. Similar remarks as those given under Figure 4.5 apply.



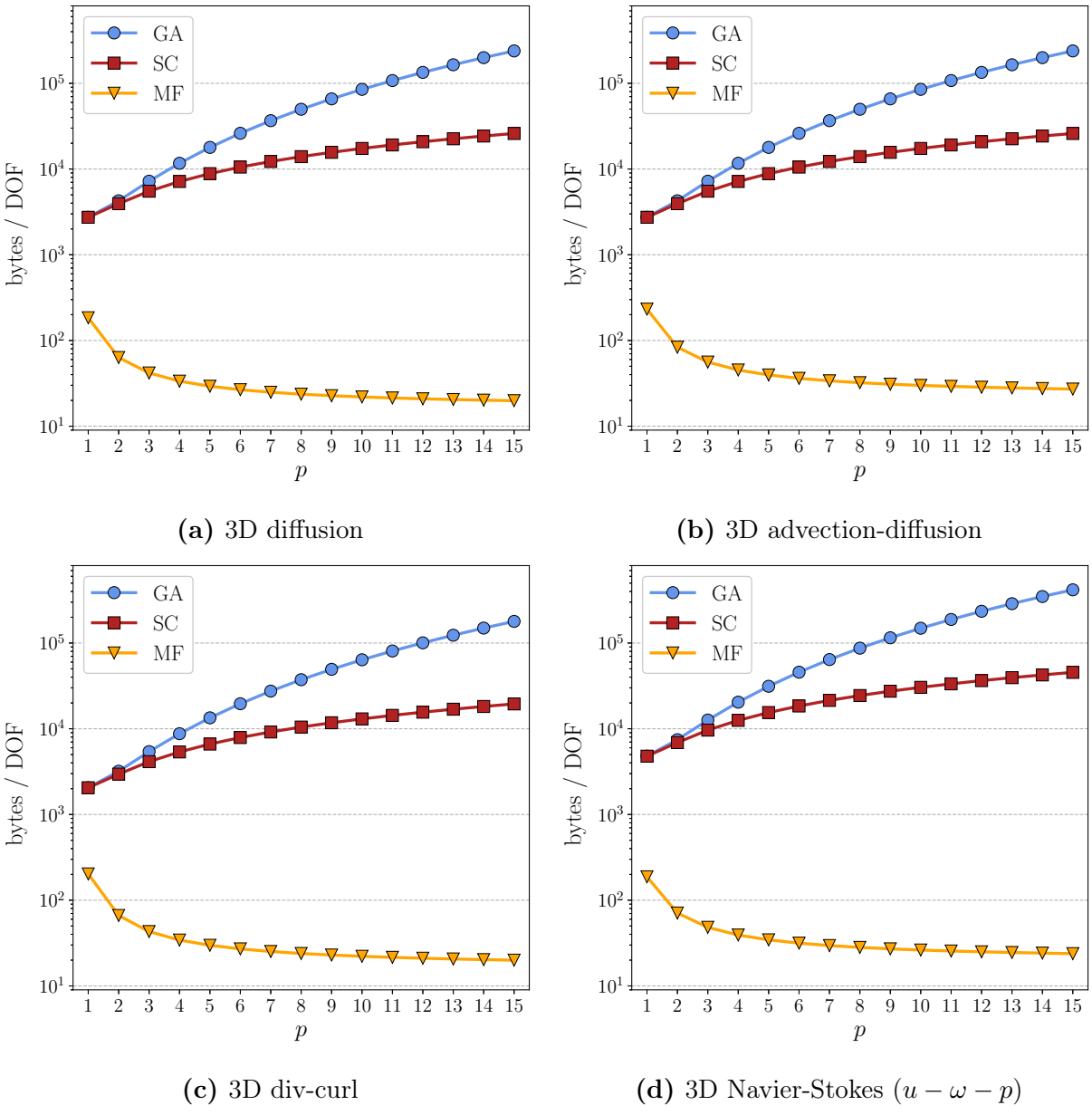
(a) 2D diffusion

(b) 2D advection-diffusion

(c) 2D div-curl

(d) 2D Navier-Stokes ($u - \omega - p$)

Figure 4.7. Comparison of memory transfers per DOF required to evaluate the least-squares operators associated with different 2D governing equations for varying element orders using the strategies described in this chapter: global assembly (GA), static condensation (SC), and matrix-free (MF) – both matrix-free approaches have the same memory access pattern. Reported values do not include structures whose access is amortized across multiple elements, such as $\mathbf{H}_A$, $\mathbf{\Psi}$, etc.



(a) 3D diffusion

(b) 3D advection-diffusion

(c) 3D div-curl

(d) 3D Navier-Stokes ($u - \omega - p$)

Figure 4.8. Comparison of memory transfers per DOF required to evaluate the least-squares operators associated with different 3D governing equations. Similar remarks as those given under Figure 4.7 apply.

Chapter 5

Software Implementation and Computational Considerations

This chapter describes L3STER – the code employed to implement the numerical methods and algorithms heretofore described in the present work. Using this numerical tool, the performance of the different operator evaluation strategies is compared experimentally. Finally, the scalability of the framework is investigated.

5.1 L3STER

While there currently exist many mature spectral/*hp* element libraries, notably those cited in the introduction, the author is not aware of any codes dedicated specifically to the least-squares formulation. To fill this gap, and to fully leverage the flexibility offered by the LSFEM, a new framework was developed. This code, named L3STER[1], should be seen as an integral part of the present work. The software is open-source, and can be found on Github [30] under the GPL 3 license. At the time of writing, L3STER is available at version 0.3.0.

Before moving on to the technical aspects of the code, let us first state the principles guiding its design. These have been enumerated below.

Flexibility and extensibility. As previously stated, one of the biggest advantages of the LSFEM is its flexibility: no weak formulation is required, and different types of

[1]The name L3STER is derived from **L**east-**S**quares **S**calable **S**pec**T**ral/*hp* **E**lement F**R**amework, and should be pronounced like the English cheese hailing from Leicestershire.

equations can be freely mixed within one system. This aspect is often mentioned in the literature; however, few concrete recommendations are provided. The first goal of the software design was therefore to enable the user to easily define their own equations (and the corresponding boundary conditions), and not lock them into a predefined set of physics. In keeping with this spirit, the decision was made to develop the code as a library, providing the requisite facilities to construct an algebraic system from a user-provided set of equations. This approach, also used in, e.g., MFEM, stands in contrast to projects such as Nektar++, where the code is compiled into binaries corresponding to different solvers, which take configuration files as their input. An additional benefit of this design choice is the ease with which L3STER can be integrated into broader frameworks (e.g. optimization) – inputs and results can be respectively provided and accessed directly via the API, no intermediate compatibility layers or conversion between file formats is necessary.

Performance. Thanks to the favorable computation-to-memory ratio afforded by the sum-factorization technique, discussed in the previous chapter, the spectral/*hp* element method offers excellent performance on modern hardware. The second design pillar was therefore to fully leverage this advantage, and aim for a high computation throughput. This consideration is made all the more pressing by the potentially increased numerical cost of the LSFEM (compared to the Galerkin method), stemming from the higher quadrature orders required to accurately evaluate the integrals involved. It should be noted that this design goal may come in tension with allowing the user to define their own operator. This issue is addressed below.

Scalability. An important aspiration for the code was for it to be able to handle large-scale simulations in three dimensions. Such a setting necessitates the use of computing clusters (supercomputers), where the computations must be performed under a distributed memory model. Support for the Message Passing Interface [67] (MPI) was therefore a foundational assumption.

Let us now focus on the more technical details of the code. L3STER is written in C++ 23, and leverages many of the features introduced in the newer standards of the language [48]. The code uses a hybrid model to achieve parallelism (sometimes called MPI+X). The primary mechanism for scalability is MPI; however, within an MPI process multi-

threading can further be employed. To this end, the Intel Threading Building Blocks (TBB) library is used. TBB has found success in other finite element codes [105], and poses an attractive alternative to runtimes such as OpenMP due to its composability.

In L3STER, equations and boundary conditions are specified using so-called kernels, which define the constituent $\mathbf{A}_i$ matrices along with the right-hand side vector $\mathbf{f}$, as described in the discussion surrounding (4.3). These kernels are invoked in the heart of the sum-factorization algorithm, at every quadrature point, and their efficiency – as well as the efficiency of the algebraic operations (chiefly matrix multiplication) immediately surrounding their evaluation – is critical to the overall performance. In order for these algebraic operations to be evaluated at high efficiency, the size of the matrices involved must be known at compile time. If it is not[2], only a generic matrix-matrix implementation can be used (such as the `dgemm` function from the BLAS – Basic Linear Algebra Subprograms), which is well-known to perform poorly for small matrix sizes. This matter is addressed in the context of spectral/*hp* elements in [59]. To directly illustrate this point, a minor experiment was carried out, which consisted of multiplying small square matrices. The multiplication was carried out using Eigen, a popular algebra library capable of heavily optimizing around compile-time-known dimensions. The results have been shown in Figure 5.1. It is clear from the plot that for very small matrices, such as the $\mathbf{\Psi}^{\pm}$ matrices involved in sum-factorization sweeps, especially at low p, providing the dimensions during compilation can speed up the computation even by orders of magnitude.

Given that performance was cited as a foundational design goal, the issue of compile-time-sized operations needed to be addressed in L3STER. To this end, all element-level parameters, primarily the approximation order and the operator parameters listed in Table 4.2, were required as compile-time constants with the help of the template C++ language feature. A reader passingly familiar with C++ templates may recoil at this statement, given that historically this feature has been seen as introducing significant complexity, and being cumbersome and "expert-only" – clearly characterizations contradictory to the goal of accessibility and ease of use stated at the outset. However, with

[2]Optimizing dynamically-sized multiplication kernels for small matrices is not impossible, but usually relies on a just-in-time (JIT) compilation approach. Libxsmm is a popular library used for this purpose. The reader may refer to the source cited in the text for a fuller discussion of this issue. In the present work, JIT-based approaches were not considered, due to the complications in the program model they impose.

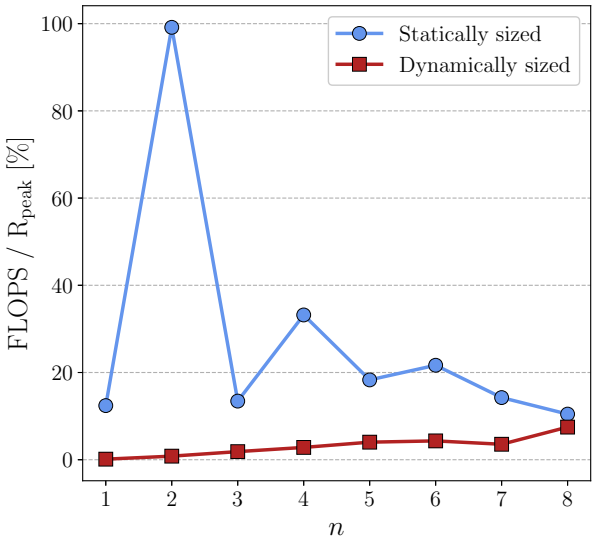


Figure 5.1. Comparison of the performance of statically and dynamically sized matrix-matrix multiplication (BLAS `dgemm`) of $\mathbb{R}^{n\times n}$ square matrices for varying values of n. Values are reported as a percentage of the theoretically achievable floating-point operations per second. Results were obtained using Eigen 3.4 on the Intel i7-10700K CPU with a fixed clock rate of 3.8GHz. The measurements were carried out using the Google Benchmark framework [35].

the advent of modern C++ features such as generic lambdas, structured bindings, and loosened requirements on the types of admissible non-type template parameters, this complexity can now be entirely hidden from the user. Indeed, the aforementioned kernel parameters can be passed as a simple structure, and the types involving them need not be mentioned in user code. To illustrate this point, the definition of the two-dimensional advection-diffusion kernel with a variable advection velocity, as specified in Equation 2.4, has been shown in Listing 5.1. Note that all physics are contained within the `AD_kernel` object, which is further imbued with the compile-time dimensions discussed above. To change them from advection-diffusion to, e.g., the Maxwell equations, only a few lines of code need to be modified. The `u` and `v` values representing the x and y components of the advection velocity, respectively, are computed using an external field, specified elsewhere in the simulation code. However, their number, which contributes to the dimensions of the algebraic structures involved, is passed at compile time under `kernel_params.n_fields`. The derivatives of the external fields (in this case unused) are available in `field_ders`. If desired, the spatial components can be retrieved from `x`. Boundary conditions can be de-

fined in the same way, the only difference being that the input decomposes into 4 elements instead of 3, where the final one contains the boundary normal vector. Finally, it should be emphasized that kernels defined in this way are free to interact with outside objects (note the ampersand in the capture group of the lambda), giving maximum flexibility to the user.

Outside of the element-level operations discussed above, L3STER uses traditional dynamically-sized algebra. Due to scalability considerations, the data structures involved – primarily the solution and right-hand side vectors, as well as the stiffness matrices when assembled approaches are used – must be distributed among the processes participating in the computation. This is achieved with the help of the Trilinos library [103], chiefly its Tpetra package. The mesh is also stored in a distributed way. The partitioning – meaning the division of the mesh among the processes – is computed using the METIS graph partitioning software [55]. METIS is widely used and allows for the computation of efficient partitionings which distribute the elements evenly, and which minimize the interfaces between the partitions, consequently reducing the communication volume. It should be noted that the communication algorithm used during matrix-free operator evaluation was implemented natively. It was determined that an asynchronous data exchange model would be preferable to that offered by Tpetra, where the shared entries must be sent and received as part of a blocking, non-overlapping phase of the evaluation procedure. Instead, in L3STER, each processes' mesh partition is further subdivided into an "interior" part, which contains all elements such that their constituent DOFs are owned by the current partition, and a "border" part, which contains elements requiring communication to read the values at the corresponding DOFs and then send back the results of the local computation. At the start of operator evaluation, non-blocking communication is initiated. Immediately afterwards, the computation in the interior mesh commences, during which the communication request handles are polled for completion. As soon as this condition is detected, the computation in the border mesh is submitted for execution with a higher priority. Once the border mesh is completed, the return communication is posted, and work on the interior mesh can continue. Assuming the ratio between the interior and border sub-partitions is sufficiently high (i.e., that we are far away from the strong scaling limit), this approach allows the communication to be fully overlapped with the computation, without the need for any intermediate synchronization.

```
constexpr auto kernel_params = KernelParameters {
  .dimension    = 2,
  .n_equations = 4,
  .n_unknowns  = 3,
  .n_fields     = 2,
};
const auto AD_kernel = wrapDomainEquationKernel<kernel_params>(
  [&](const auto& input, auto& output) {
    const auto& [field_vals, field_ders, x] = input;
    const auto& [u, v]                      = field_vals;
    auto&       [operators, rhs]            = output;
    auto&       [A0, Ax, Ay]                = operators;
    const auto  lambda                      = 1.;
    const auto  source                      = 1.;

    A0(1, 1) = -1.;
    A0(2, 2) = -1.;

    Ax(0, 0) = u; Ax(0, 1) = -lambda;
    Ax(1, 0) = 1.;
    Ax(3, 1) = 1.;

    Ay(0, 0) = v; Ay(0, 2) = -lambda;
    Ay(2, 0) = 1.;
    Ay(3, 2) = -1.;

    rhs[0] = source;
  }
);
```

Listing 5.1. Definition of the 2D advection-diffusion operator in L3STER.

5.2 Computational Study of Operator Evaluation Performance

In order to determine the peak possible performance of the implementation described above, the following computational experiment was carried out. The operators listed in Table 4.2 were evaluated in a single quadrilateral or hexahedral element (depending on the spatial dimension of the operator) of varying order p. Three matrix-free evaluation strategies were employed: the local element approach, and the sum-factorization technique with and without odd-even decomposition. For operators requiring external fields (e.g. the advection velocity), the nodal values of these fields were provided in an external array, meaning that the work associated with copying and transposing these values was performed. However, only the data for the single element were stored. Given the above, the working set of the computation was able to be fully cached by the CPU. This idealized setting allowed for a clearer assessment of the efficiency of the generated machine code, without the results being influenced by main memory bandwidth and latency considerations. The execution times were measured using Google Benchmark [35], a widely-used micro-benchmarking library. The computations were run on a single core of the Intel i7-10700K CPU, at a fixed clock rate of 3.8GHz. The code was compiled using GCC 14.2 with the `-O3 -march=native -mtune=native` flags, meaning that full optimization was enabled and the compiler was free to emit AVX2 vector instructions.

The results have been reported in Figure 5.2 and Figure 5.3. The DOF/s metric was obtained by extrapolating the time measured for a single element to an infinite Cartesian grid. In order to maintain the idealized setting of the study, since the performance of the assembled approaches is strictly memory-bound and cannot be meaningfully assessed using a 1-element mesh, the green lines showing the performance of the global assembly approach with static condensation were not obtained experimentally, but instead determined based on the quotient of the main memory bandwidth of the test system – 15 GB/s – and the operators' memory requirements outlined in the previous chapter. The global assembly approach without static condensation is omitted from the plots for clarity.

It is evident from the results that for quadrilateral and hexahedral elements the local element approach is strictly inferior to the other strategies, even for low values of p. It should be emphasized that this may not always be the case for other element types not

considered in the present work (notably triangles and tetrahedrons), as has been reported in, e.g., [11, 107]. In two dimensions, the performance of the assembled approach was comparable to that of the remaining two matrix-free strategies for the less computationally intensive operators, namely diffusion and div-curl. For $p < 3$, static condensation was more efficient, while at higher approximation orders the matrix-free approaches were faster. However, for the advection-diffusion and Navier-Stokes operators, static condensation was more performant for all p considered. In three dimensions, the efficiency of the assembled approach drops rapidly as p increases, and the matrix-free approaches quickly become superior. The break-even points at which the matrix-free approach was at least as efficient as the assembled strategy have been summarized in Table 5.1.

Operator	Break-even p	
	Quad	Hex
Diffusion	3	2
Advection-diffusion	-	3
Div-curl	3	1
Navier-Stokes ($u - \omega - p$)	-	1

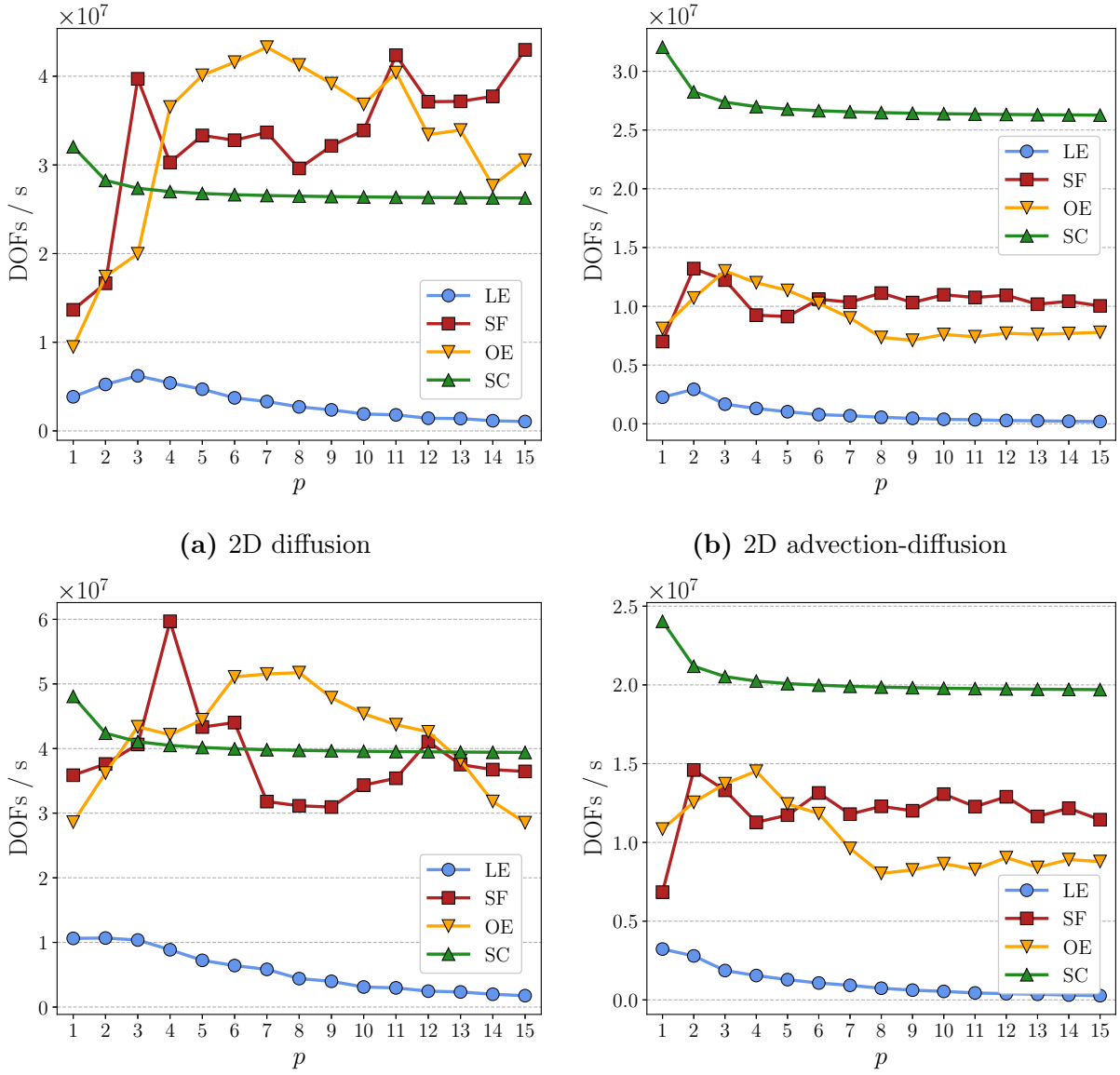
Table 5.1. Single-threaded operator evaluation performance study: element orders at which the sum-factorization technique performs better than static condensation.

The final aspect of the results which must be addressed is the performance impact of employing the odd-even decomposition. As was discussed in the previous chapter, leveraging this technique formally reduces the number of floating-point operations required for executing a sum-factorization sweep by nearly half. However, it implies additional data movement, which was discounted in the previous analysis. Furthermore, as the relevant array sizes are halved, it is more difficult to optimally vectorize the algebraic operations. This behavior is particularly visible in the diffusion and div-curl plots (in both two and three dimensions) for $p \leq 3$. When the $\mathbf{\Psi}^{\pm}$ matrices only have one or two rows and columns, the compiler cannot emit SIMD instructions which fully saturate the vector registers (in the AVX2 instruction set architecture, the maximum vector register size is 256 bits, meaning that it can hold 4 double-precision floating-point values). Another

interesting effect occurs at $p = 8$ for the advection-diffusion and Navier-Stokes operators. Examining the disassembled machine code revealed that at this order Eigen falls back on the generic matrix-matrix multiplication implementation, and the generated assembly ceases to benefit from full inlining and loop unrolling, causing a noticeable drop in the throughput. When standard sum-factorization is used, this threshold is exceeded earlier, and the performance stabilizes around $p = 5$. Nevertheless, in all cases there exists an intermediate regime where using the odd-even decomposition provides an increase in efficiency. Overall, the present study confirms that a spectral/hp element code should support all evaluation strategies. While some emergent trends are clear, notably the excellent performance of sum-factorization in three dimensions, different operators may benefit from different strategies at different approximation orders.

As mentioned at the outset, the experiment described above was meant to establish a performance ceiling for the matrix-free approaches, isolated from memory latency and bandwidth considerations. It was carried out in a highly conceited setting, with only one element handled by a single core. Before moving on to discussing the scalability of L3STER, it is of interest to investigate its performance under real-world conditions. To this end, a second study was undertaken. This time, only the advection-diffusion operator was considered. As we will establish in the next chapter, the advection-diffusion step carries the majority of the cost of the splitting scheme described in section 2.4, and the conclusions of the present study can be generalized to other equations. Here, to more meaningfully assess the operator evaluation strategies, an advection-diffusion problem was solved in fully periodic square and cubic domains for varying approximation orders. The domains were discretized using an isotropic Cartesian grid, and the number of elements at each p was chosen so that the resulting number of nodes would be no less than 46,656. The algebraic problem was solved using the conjugate gradient method with a Jacobi preconditioner, and the duration of the iterations was recorded. Note that the measurement included not just the operator evaluation, but also the application of the preconditioner and other auxiliary operations (e.g. dot products) performed by the conjugate gradient solver. The computation was carried out on the same machine as before; however, this time, all 8 physical cores were used within 1 MPI process.

The results have been shown in Figure 5.4. In order to compare per-core performance, the measurements are reported as the number of conjugate gradient iterations performed,



(a) 2D diffusion

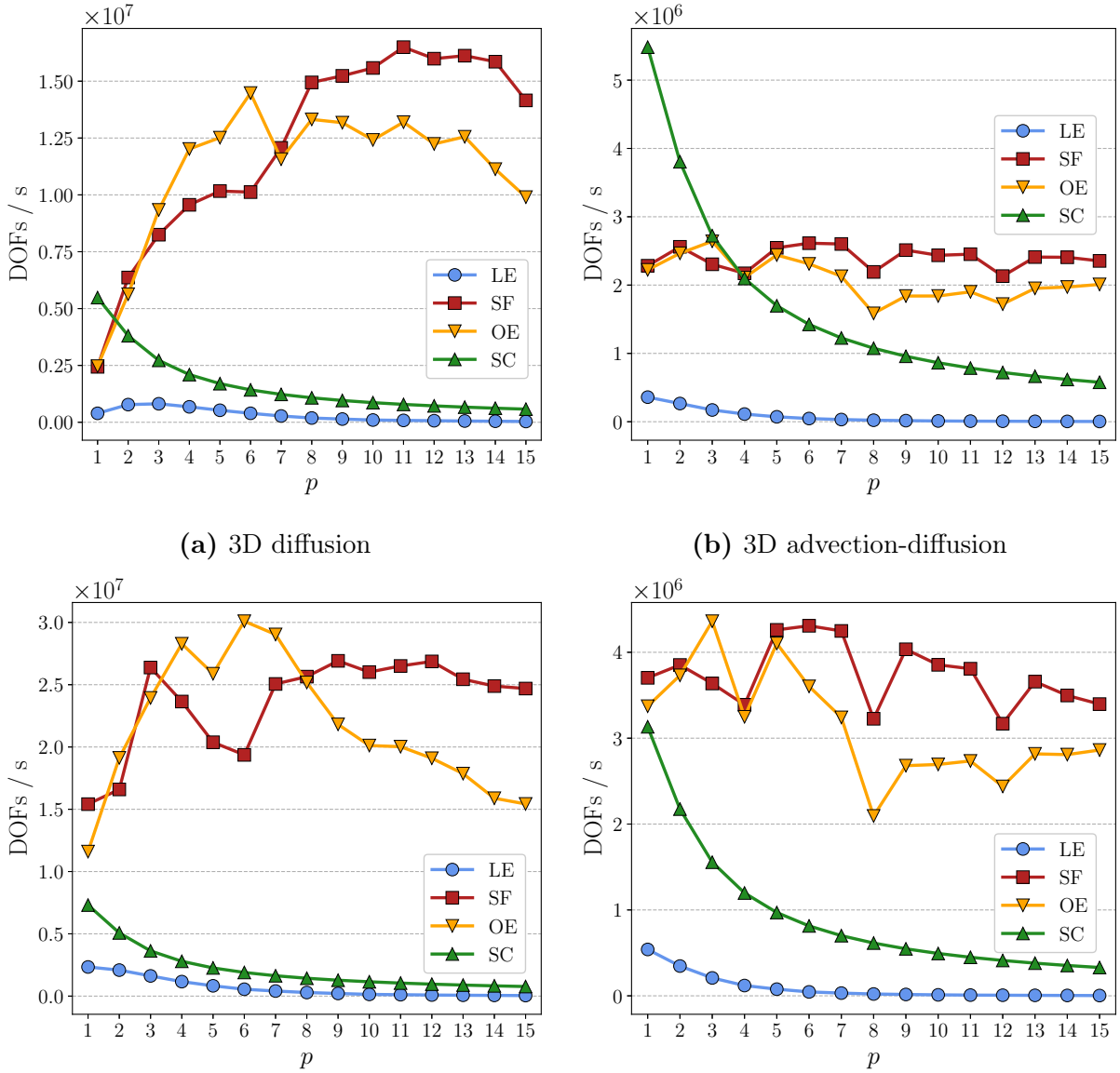
(b) 2D advection-diffusion

(c) 2D div-curl

(d) 2D Navier-Stokes ($u - \omega - p$)

Figure 5.2. Comparison of single-threaded evaluation performance of least-squares operators associated with different 2D governing equations for varying element orders using the different evaluation strategies: local element (LE), sum-factorization (SF), sum-factorization with odd-even decomposition (OE), and global assembly with static condensation (SC).

multiplied by the number of degrees of freedom, and divided by the product of the elapsed wall time and number of CPU cores. A similar metric was used in [28]. The trends in the results are similar to those observed in the idealized test; however, some clear quantitative differences exist. It is immediately clear that in the more representative test, the iteration speed is not as high as in the idealized setting. The degradation in



(a) 3D diffusion

(b) 3D advection-diffusion

(c) 3D div-curl

(d) 3D Navier-Stokes ($u - \omega - p$)

Figure 5.3. Comparison of single-threaded evaluation performance of least-squares operators associated with different 3D governing equations for varying element orders using the different evaluation strategies: local element (LE), sum-factorization (SF), sum-factorization with odd-even decomposition (OE), and global assembly with static condensation (SC). Note the difference in the exponents on the y axes between the plots on the left and right.

performance is particularly significant in the assembled approach. Consequently, the gap between static condensation and the matrix-free strategies in two dimensions narrows. This effect is due to the memory-bound nature of assembled approaches – the memory bandwidth does not scale with the number of CPU cores. It should be emphasized that

this is *a fortiori* true on high-performance computing (HPC) machines, where the ratio of computing power to memory bandwidth is typically greater than on desktop machines, such as the one used here.

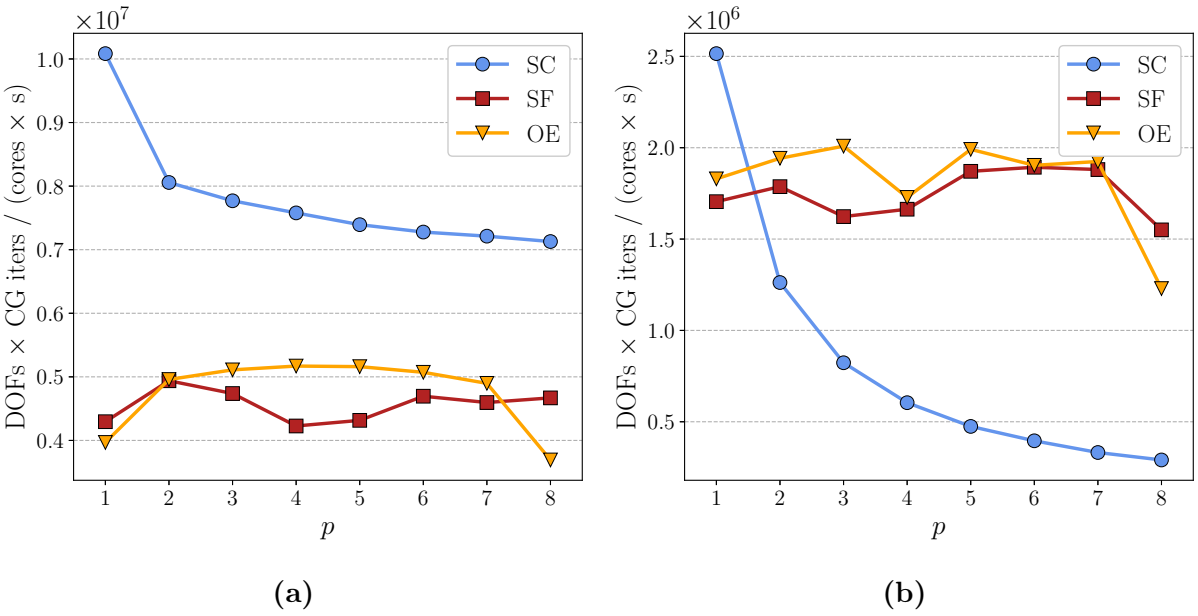


Figure 5.4. Comparison of advection-diffusion operator evaluation in (a) 2D and (b) 3D using different strategies: static condensation (SC), sum-factorization (SF), and sum-factorization with odd-even decomposition (OE).

5.3 Scaling Benchmarks

The final issue which will be addressed in this chapter is the scalability of the developed code. To investigate this aspect, the three-dimensional advection-diffusion equation in a periodic cube domain was used as the sample problem. Once again, the conjugate gradient method with a Jacobi preconditioner was used as the algebraic solver, and the sum-factorization technique with odd-even decomposition was employed to evaluate the operator. The approximation order was fixed at $p = 3$.

The sample problem was solved using varying numbers of nodes and discretization sizes. Both strong and weak scaling tests were performed. In the former, the problem size was fixed at 967,680 DOFs in one run, and 15,482,880 DOFs in the second run, and the number of nodes was systematically increased. This setting reflects the more typical use-case of HPC machines, where a simulation must be performed at a given resolution,

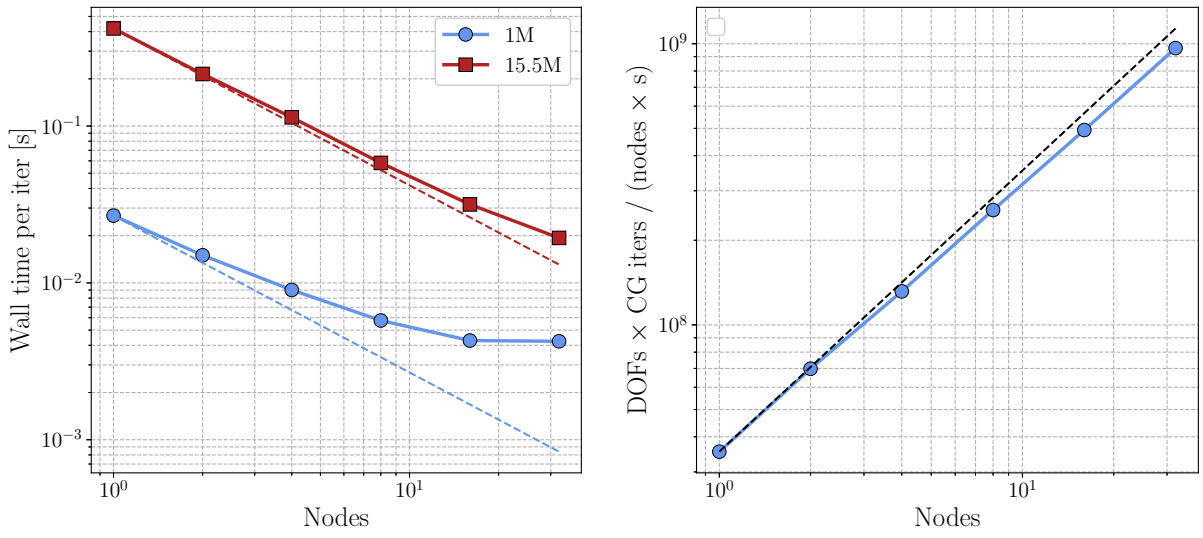
and the result is desired as quickly as possible – more resources are used to speed up the computation. In the weak scaling test, the number of DOFs per node was fixed at 967,680, and the overall problem size grew proportionally to the computational resources employed.

The tests were carried out on the Topola cluster at the Interdisciplinary Center for Mathematical and Computational Modeling of the Warsaw University (ICM UW). A description of the hardware available on the nodes of the cluster has been provided in Table 5.2. The nodes are networked together using the Mellanox InfiniBand interconnect. Each process was assigned to a single CPU (socket), and multi-threading was used to distribute the work between its cores. This approach leverages the hybrid parallelism model described above.

CPU model	Intel Xeon E5-2697 v3
Clock rate	2.6GHz (dynamic scaling enabled)
Microarchitecture	Haswell
Available vector instruction set	AVX2 (256b)
Cores per CPU	14
CPUs per node	2
Memory on node	64/128GB

Table 5.2. Topola cluster node specification.

The result have been shown in Figure 5.5. In the weak scaling test, the code can be seen to scale near-perfectly up to 32 nodes. In the strong scaling test, it is possible to observe the strong scaling limit for the problem sizes considered. In the smaller simulation, the point of diminishing returns is already reached at around 2 nodes. By the time 32 nodes are employed, the computation ceases to scale altogether. However, in the larger problem, good strong scaling behavior can be observed up to 16 nodes, after which the curve begins to flatten out. The strong scaling limit in this experiment can therefore be deduced to be approximately 34 thousand DOFs per CPU core.



(a) **(b)**

Figure 5.5. Scaling tests: (a) strong scaling for approx. 1 million and 15.5 million total DOFs, (b) weak scaling for approx. 1 million DOFs per node. Dashed lines mark ideal scaling.

Chapter 6

Applications

This chapter contains examples of applications of the methods and algorithms discussed in the present work. First, the efficacy of the splitting scheme is investigated. Next, the example of the Taylor-Green vortex is used to compare its performance with the standard coupled approach. Subsequently, three-dimensional flow around a cylinder is simulated, showcasing a more computationally demanding application. Finally, fracture flow is considered, providing a more geometrically complex test case.

6.1 Splitting Scheme Convergence Study

In this study, we seek to empirically establish the convergence properties of the splitting scheme described in section 2.4 using the manufactured solution approach. To this end, the following velocity and pressure fields were assumed:

$$u(x, y, t) = 2\sin(\pi x)\cos(\pi y)\sin(t), \tag{6.1a}$$

$$v(x, y, t) = -2\cos(\pi x)\sin(\pi y)\sin(t), \tag{6.1b}$$

$$p(x, y, t) = 2\sin(\pi x)\sin(\pi y)\cos(t). \tag{6.1c}$$

Note that $\mathbf{u} = [u, v]^T$ is solenoidal by construction and hence fulfills the incompressibility constraint. Next, the source terms $\mathbf{f}$ and $\mathbf{g}$ were specified so that (6.1) fulfills the equations (2.8a) and (2.9b) in the domain and on the open boundary, respectively. The domain was set to the square $\Omega = [0, 2] \times [-1, 1]$, the time bound to $T = 0.2$, and the viscosity to $\nu = 0.01$. The setting described above is taken directly from [24], similar test cases are used elsewhere in the literature. Defining the solution (6.1) *a priori* allows for an exact

evaluation of the error obtained in numerical simulation.

In order to identify the effects of the inclusion of the open boundary condition in the scheme originally proposed in [83], two sets of simulations were run. In the first set, Dirichlet BCs for the velocity were imposed on the entire boundary, i.e., $\Gamma_O = \emptyset$. In the second, open boundary conditions were imposed on the right boundary and the right half of the top boundary, i.e., $\Gamma_O = \{(x, y) : x = 2 \wedge -1 \leq y \leq 1\} \cup \{(x, y) : 1 \leq x \leq 2 \wedge y = 1\}$. In both cases, the domain was discretized using 4 square spectral elements. The order of the time stepping scheme was set to $J = 2$.

In the spatial convergence test, the time step size was fixed at $\Delta t = .001$ (200 time steps), and the element order was varied from $p = 2$ to $p = 14$. The resulting L_2 error norms for the velocity, pressure, and vorticity have been shown in Figure 6.1(a) and Figure 6.1(b). It can clearly be seen that for both boundary configuration considered, the scheme exhibits spectral (exponential) convergence in space up to around element order $p = 10$, where the plots level off due to the error associated with time discretization becoming dominant. As can be expected, when open BCs are present the error is slightly higher, since these are inherently "weaker" than Dirichlet BCs. Furthermore, at $p = 14$, the pressure and vorticity errors in the open BC test exhibit an uptick, not present in the test with strictly Dirichlet BCs. This effect cannot be readily explained by the author, however, it should be noted that the increase in error is only minor, and that $p = 14$ lies outside the range of element orders of practical interest.

In the time convergence test, the element order is fixed at $p = 12$, and the time step size is varied from $\Delta t = 0.025$ to $\Delta t = 0.0001953125$, corresponding to $2^3, \ldots, 2^{10}$ time steps. The results have been shown in Figure 6.1(c) and Figure 6.1(d), along with a dashed reference line marking the desired slope. It is evident from these plots that for both boundary configurations considered, the scheme exhibits second order convergence in time. Since $J = 2$ was used, this result indicates optimal temporal convergence. Interestingly, this conclusion stands in conflict with the original work by Pontaza [83], where the errors were found to converge at approximately $\mathcal{O}(\Delta t^{2.8})$, not $\mathcal{O}(\Delta t^2)$. However, the setting used in the cited work was slightly different: the domain was taken to be the unit square, and the manufactured solution (6.1) involved a different combination of trigonometric functions. When this setting was replicated by the author of the present work, the results were found to be in complete agreement with [83]. It seems therefore

that the "enhanced" rate of 2.8 may have been an artifact of the specific example being considered, and not an inherent feature of the combination of the splitting scheme with the LSFEM, as hypothesized by Pontaza. While this conclusion is disappointing, it should be emphasized that the strategy in question is nevertheless of great interest, even if it is "only" optimal.

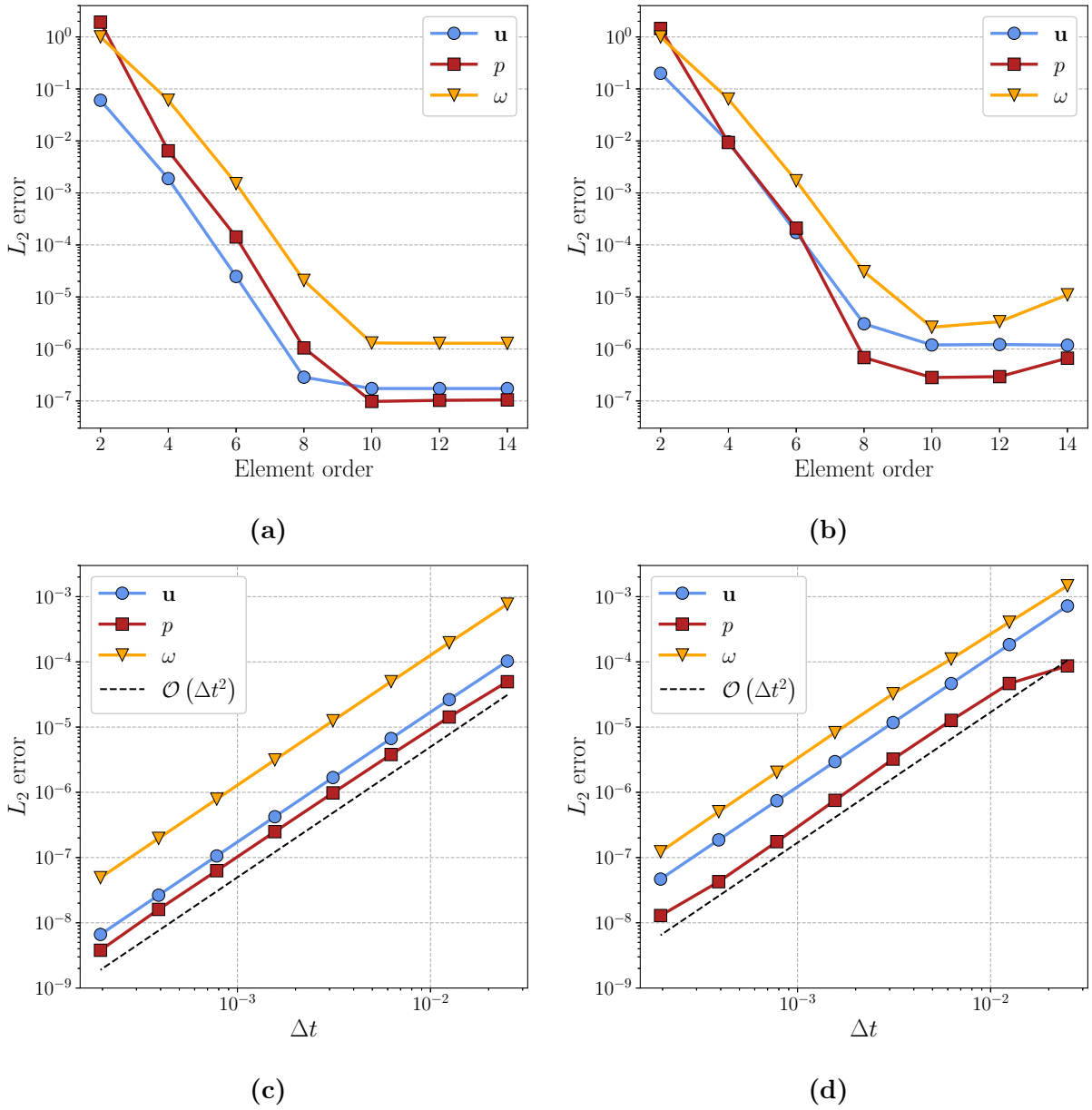


Figure 6.1. Manufactured solution study: (c), (d) space and (a), (b) time convergence rates for solution components computed using only Dirichlet BCs (left), and Dirichlet and open BCs (right).

6.2 Taylor-Green Vortex

The Taylor-Green vortex is a classical two-dimensional solution to the Navier-Stokes equations, dating back to 1937 [102]. It describes a vortex in a bi-periodic square domain, which decays exponentially in the absence of external forces. Over the years, efforts have been made to generalize this solution, culminating in the recent work by Antuono [2], where the author derived a fully three-dimensional, tri-periodic vortex. This result[1], applied to the tri-unit cube domain $\Omega = [0,1] \times [0,1] \times [0,1]$, has been given below:

$$u(\mathbf{x},t) = A(t)\left[\sin(x'-\alpha)\cos(y'-\beta)\sin(z') - \cos(z'-\alpha)\sin(x'-\beta)\sin(y')\right], \quad (6.2a)$$

$$v(\mathbf{x},t) = A(t)\left[\sin(y'-\alpha)\cos(z'-\beta)\sin(x') - \cos(x'-\alpha)\sin(y'-\beta)\sin(z')\right], \quad (6.2b)$$

$$w(\mathbf{x},t) = A(t)\left[\sin(z'-\alpha)\cos(x'-\beta)\sin(y') - \cos(y'-\alpha)\sin(z'-\beta)\sin(x')\right], \quad (6.2c)$$

$$p(\mathbf{x},t) = p_0 - \frac{1}{2}\left\|\mathbf{u}(\mathbf{x},t)\right\|^2, \quad (6.2d)$$

where $\alpha = \frac{5}{6}\pi$, $\beta = \frac{1}{6}\pi$, and p_0 denotes a generic constant. The amplitude $A(t)$ and coordinate transformation $\mathbf{x}'(\mathbf{x})$ are given by

$$x_i'(x_i) = \pi\left(2x_i + \frac{1}{2}\right), \quad i \in \{1,2,3\}, \quad (6.3a)$$

$$A(t) = \frac{4\sqrt{2}}{3\sqrt{3}}\, e^{-12\nu\pi^2 t}. \quad (6.3b)$$

The isocontours of the Q criterion, which show the fundamentally three-dimensional nature of the vortex, have been shown in Figure 6.2.

It should be emphasized that the difference between a manufactured solution (such as the one described in the previous section) and a "proper" solution lies in the treatment of the source terms. In the former, the right-hand side is crafted so that a preconceived solution satisfies the equations. In the latter, the solution is sought based on a given source term (in this case $\mathbf{f} \equiv \mathbf{0}$), no *a priori* knowledge is assumed.

In the present study, the generalized Taylor-Green vortex (6.2) is used as a test case to further investigate the time convergence of the splitting scheme described in section 2.4 in three dimensions. Moreover, the performance and accuracy of the splitting scheme is compared with that of the un-split approach, wherein the system (2.13) is directly

[1] In the original work [2], the solution is more generally defined in terms of the wave number k, related to the cube side length L by $L = \frac{2\pi}{k}$, and a translation parameter R. The velocity components in (6.2) are provided for $L = 1$ ($k = 2\pi$) and $R = 0$.

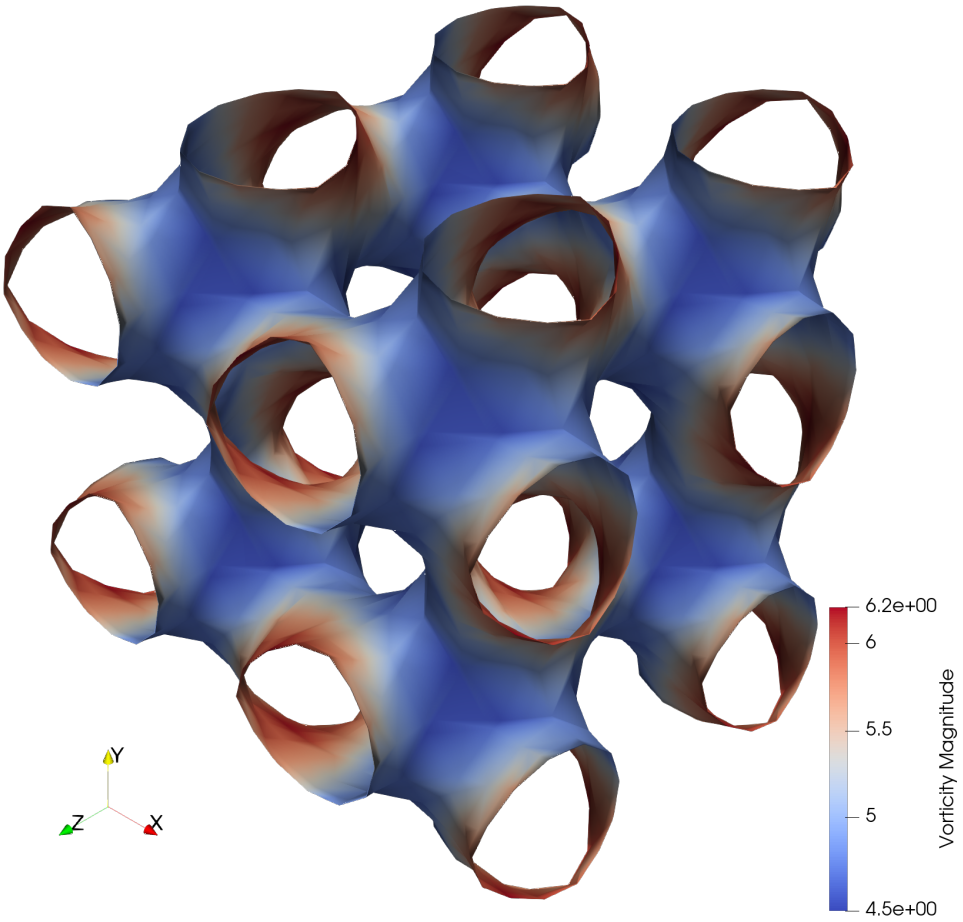


Figure 6.2. Generalized Taylor-Green vortex: isocontours of the Q-criterion ($Q = 1$) colored by vorticity magnitude at $t = T$.

integrated in time. To this end, simulations employing both strategies were carried out for varying time step sizes. Despite the comment made above, the domain was not made tri-periodic, but instead Dirichlet boundary conditions based on (6.2) were imposed on the velocity. This setting corresponds to the manufactured solution approach (albeit with no source terms). Performing the simulations in a fully periodic domain was precluded by the second and third phases of the splitting scheme. In the absence of Dirichlet boundaries, the solution of the div-curl equation is unique only up to a constant. Similarly, in this setting the pressure Poisson problem is well-posed only if the integral of the right-hand side $\nabla^T \mathbf{u} : \nabla \mathbf{u}$ over the domain is zero. The former may be addressed, at least partially, by imposing an additional Dirichlet condition $\mathbf{u} = \hat{\mathbf{u}}$ at a single point in the domain. However, when performing the simulation using the splitting scheme was attempted, the pressure Poisson problem would fail to converge. This may have been caused by the numerical error in the velocity field causing the aforementioned integral to depart from

zero, although further investigation is needed to confirm this hypothesis. The question of formulating the splitting scheme in a fully periodic domain is left open for future research. It should be noted that this problem does not affect the un-split approach, which performed as expected in the tri-periodic case.

The domain was discretized using a grid of 2 elements in each dimension, and the approximation order was set to $p = 10$. For time discretization, 2^n time steps were used, for values of n in the range $2 \leq n \leq 6$. The order of the time stepping schemes was chosen to be $J = 2$. The viscosity was set to $\nu = 0.01$, and the time horizon of the simulation to $T = 0.585$, corresponding to approximately one half-life of the vortex. In the un-split scheme, in order for the comparison to be meaningful, only one Newton iteration was performed at each time step. All algebraic problems were solved using the conjugate gradient method, where the absolute convergence threshold was set to 10^{-7}. The sum-factorization technique was used for operator evaluation. Care was taken to provide the best possible initial guesses for the solutions, which were obtained via time-extrapolation, or – in the case of the div-curl step of the splitting scheme – set to the intermediate velocity approximation $\hat{\mathbf{u}}$. Although this issue is not quantitatively investigated in the present work, it should be stated that properly seeding the conjugate gradient solver noticeably (though not dramatically) reduced the number of iterations needed for convergence. The computations were carried out on the Intel i7-10700K CPU, using 8 physical cores with a fixed clock rate of 3.8GHz.

The results of the simulations have been shown in Figure 6.3. The un-split approach exhibits $\mathcal{O}(\Delta^2)$ convergence in the L_2 errors of both solution components for the first three time step sizes considered, after which the error in the pressure levels off, while the slope of the velocity error remains unchanged. Conversely, the splitting scheme achieves a convergence rate of $\mathcal{O}(\Delta^{2.8})$ in the velocity error. This result indicates that the findings of [83] discussed previously may not, after all, have been incidental. On the other hand, the slope of the pressure error is somewhat inconclusive, but seems less than $\mathcal{O}(\Delta^2)$. Overall, the less conclusive nature of these results, compared to the example from section 6.1, is likely a consequence of the less contrived setting of the present study. Nevertheless, the splitting scheme is shown to be convergent. Its accuracy is comparable to the un-split approach, for the lowest time step size considered it can even be said to exhibit smaller errors.

The number of conjugate gradient iterations required for convergence has been visualized in Figure 6.3(b) using violin plots [46], which display the distribution across time steps. On average, each of the phases of the splitting scheme requires significantly fewer iterations than the un-split equations, and furthermore exhibits a much narrower spread between time steps. The number of iterations per phase monotonically decreases with the time step size, whereas in the un-split case, this relationship is not as clear. This behavior can be attributed to the saddle-point nature of the Navier-Stokes system. It can therefore be stated that the splitting scheme's performance is more consistent.

Finally, the total wall time of the simulations has been shown in Figure 6.3(c). For the three lower time step sizes considered, the computational cost of the splitting scheme was less than that of the un-split approach. In all cases, the majority of the computational effort was spent on the advection-diffusion stage, while the cost of the div-curl phase was near-negligible. In summary, it can be stated that the splitting scheme offers comparable accuracy and better, more consistent performance than the un-split approach.

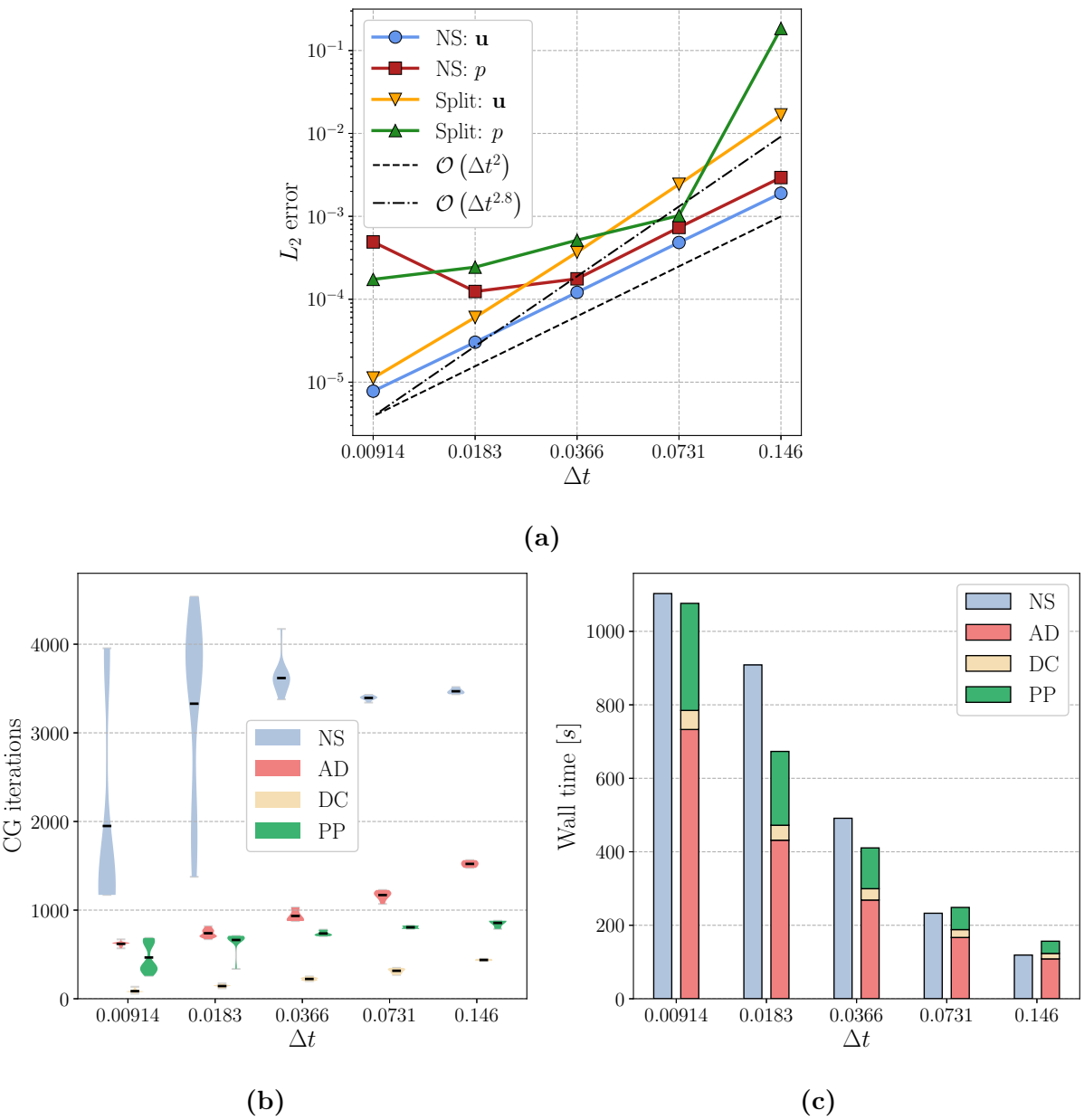


Figure 6.3. Generalized Taylor-Green vortex simulation results: (a) time convergence rates, (b) conjugate gradient iterations required for convergence (solid dashes denote the means), (c) total wall times. Iterations and wall times are reported individually for each of the problems: unsplit Navier-Stokes (NS), advection-diffusion (AD), div-curl (DC), pressure Poisson (PP).

6.3 Flow Past a Cylinder

Flow past a smooth cylinder is a classical problem in fluid mechanics, and has been studied both experimentally [93] and numerically [42] for many decades. It is well known that at $Re \sim 47$ the flow becomes unstable, leading to vortex shedding behind the cylinder (the famous von Kármán vortex street). Furthermore, at approximately $Re \sim 188$, the onset of a secondary instability causes the flow to become three-dimensional in nature. As the Reynolds number increases, the flow becomes more complex, eventually leading to a boundary layer transition at $Re \sim 2 \cdot 10^5$ [27].

In the present study, an intermediate regime of $Re = 400$ is considered, leading to rich, three-dimensional flow dynamics, but not imposing excessive requirements on the mesh resolution. A direct numerical simulation (DNS) is performed, and the flow is compared to the results of the extensive study undertaken in [51]. Such a test case allows for an evaluation of the numerical tool proposed in the present work in a real-world, computationally demanding setting.

The computational domain consisted of a box surrounding a cylinder of diameter $D = 1$, oriented along the z axis. The flow was directed along the x axis by imposing the Dirichlet condition $\mathbf{u} = [1, 0, 0]^T$ on the upstream face parallel to the yz plane. A no-slip condition, $\mathbf{u} = \mathbf{0}$, was imposed at the cylinder wall. Periodic boundary conditions were imposed on the faces perpendicular to the z (spanwise) and y (transverse) axes. Finally, the open boundary condition (2.9b) was imposed on the outlet, i.e., the downstream face parallel to the yz plane. The flow was simulated using the splitting scheme described in section 2.4, starting from a stationary fluid and the pressure field $p^0 = 0$. The time step was set to $\Delta t = 0.05$, and the problem was integrated in time until $T = 190$. The inlet velocity was gradually ramped up over the initial half time unit (10 time steps). The domain was discretized using 109,100 elements; the view of the mesh parallel to the xy plane has been shown in Figure 6.4. In the spanwise direction, 25 uniformly distributed elements were used. The approximation order was set to $p = 3$. A summary of the configuration described above, along with the exact domain dimensions and additional mesh parameters, has been provided in Table 6.1. The computations were carried out using 24 nodes of the Topola cluster at ICM UW, described in the previous chapter. The simulation required a total of approximately 70 thousand CPU hours.

Let us now discuss the results of the simulation. The history of the lift coefficient C_L

Parameter	Value
Distance from inlet to cylinder center	$10D$
Distance from cylinder center to outlet	$16D$
Distance from cylinder center to transverse boundaries	$15D$
Spanwise domain length	$5D$
Height of the first layer of the mesh surrounding the cylinder	$0.02D$
Element radial expansion ratio near the cylinder	1.135
Number of elements in the spanwise direction	25
Approximation order	3
Number of elements	109,100
Number of nodes (excluding periodic)	2,963,175
Time step	0.05
Reynolds number	400
Inlet velocity ramp-up time	0.5

Table 6.1. Flow past a cylinder: case parameters.

has been shown in Figure 6.5(a). This plot alone reveals the three phases of the flow. Initially, the lift force is zero, as the flow remains symmetric about the xz plane. This stage has been shown in Figure 6.6. Two recirculation zones can be seen forming behind the cylinder. Eventually, at approximately $t \sim 25$, the onset of the first instability can be seen, as the lift force begins to oscillate. After a brief period of "wobbling," the vortices detach, and the vortex-shedding behavior ensues. These dynamics constitute the second phase, where the flow is no longer stationary, but remains entirely two-dimensional, as the spanwise velocity component is zero in the entire domain. A snapshot of the isocontours of the z component of the vorticity has been shown in Figure 6.7 – the two-dimensional nature of the velocity field can clearly be seen. Note the slight artifacts at the outflow, which can be attributed to the open boundary condition. However, as the domain extends sufficiently far downstream, these artifacts do not meaningfully impact the upstream flow. It should also be noted that despite the slight backflow periodically appearing at the outflow boundary, the simulation experienced no stability issues.

The second phase concludes around $t \sim 110$ with the onset of the secondary instability.

The transition to three-dimensional flow dynamics has been captured in Figure 6.8(a), where a snapshot of the isocontours of the Q-criterion at $t = 117.5$ has been shown. The vortex rolls exiting the domain are still nearly-perfectly two-dimensional; however, the rolls closer to the cylinder exhibit noticeable undulations. Finally, the vortex currently being shed by the cylinder at the time the snapshot was taken has partially broken down into smaller structures, forming a visible ribcage-like pattern. Once the two-dimensional structure of the flow breaks down, the expected chaotic dynamics quickly take over. A snapshot of the fully developed wake taken at $t = 155.5$ has been shown in Figure 6.8(b). It can be seen that the once-cylindrical primary vortex rolls are now highly deformed, and streamwise-oriented detached vortical structures are present.

In order to quantitatively compare the obtained results to those available in the literature, the power spectra of the lift force in the two unstable regimes described above were computed using the Fourier transform. The results have been shown in Figure 6.5(b) and Figure 6.5(c). The first spectrum was computed in the interval $65 < t < 110$, corresponding to the two-dimensional stage, while the second was obtained using the history of the lift force for $130 < t < T$, i.e., the fully developed three-dimensional flow. The dominant frequencies f clearly visible in the plots correspond to the primary vortex-shedding behavior, and can be used to define the Strouhal number St using the relation

$$St = \frac{fD}{U}, \tag{6.4}$$

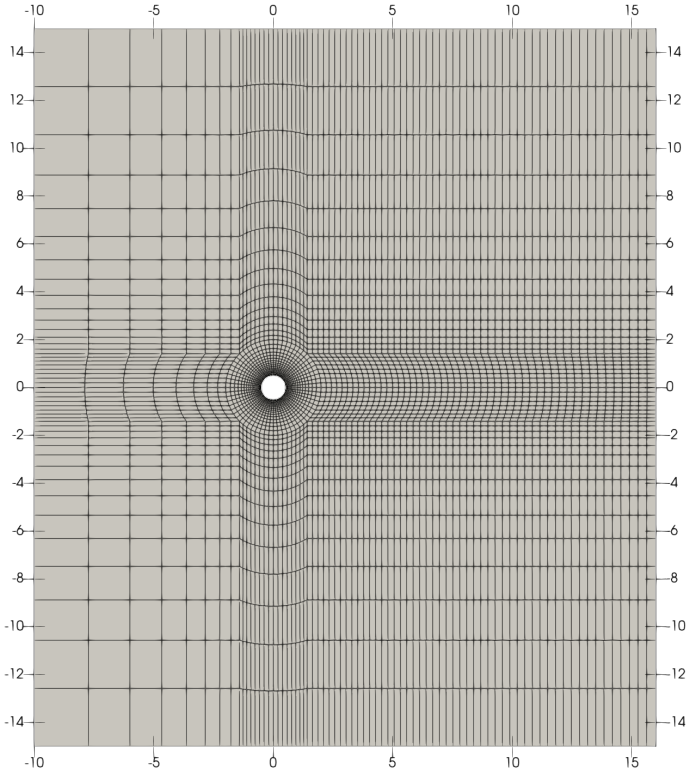
where U is the free-stream velocity, in this case equal to 1. The computed values have been compared to the figures obtained by Jiang et al. [51] in Table 6.2. The results are in excellent agreement, with the discrepancies for the two- and three-dimensional flow equal to 3.7% and 0.1%, respectively. The noticeable difference for the two-dimensional flow is likely due to the shorter window of measurement in the present work. In [51], the reported value is based on a two-dimensional DNS, and can therefore be computed over a longer term, given that no secondary instability can occur in such a setting. Overall, the behavior of the Strouhal number described in the literature is well reproduced using the current approach. The three-dimensional instability causes St to decrease, as the recirculation region extends further downstream, and the shear layer is consequently weaker at the point of shedding.

Finally, in order to provide a more in-depth comparison than the one afforded by merely examining the Strouhal number, the isocontours of the x component of the vorticity

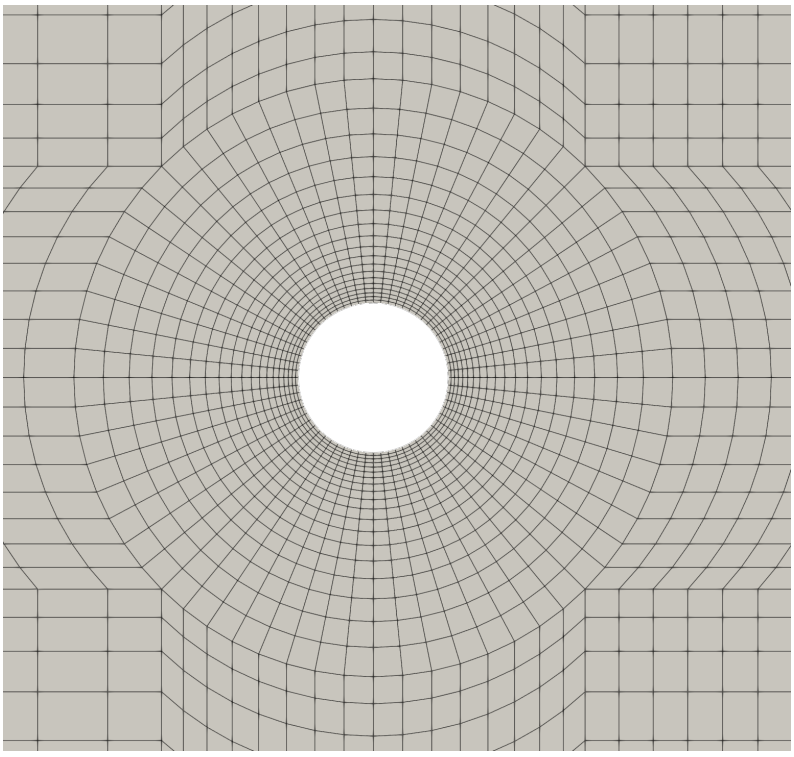
	St_{2D}	St_{3D}
Jiang et al. [51]	0.219	0.204
Present work	0.227	0.204

Table 6.2. Flow past a cylinder: comparison of the Strouhal numbers with [51]. St_{2D} is the Strouhal number in the two-dimensional regime, while St_{3D} corresponds to the fully developed flow.

in the developed flow were shown in Figure 6.9. Similar visualizations can be found in [51]. In the cited work, these results are provided for $Re = 300$; however, as is explained therein, the overall structure of the flow does not change in the range $270 < Re < 1000$, $Re = 270$ being a transition point between two different modes of the three-dimensional instability. Similarities in the flow features between the present and cited works pertain to the structure of the vortices revealed by the isocontours of ω_x. Given the sufficiently long spanwise dimension L_z of the domain, this structure is more chaotic and not as well organized as when the domain was restricted to $L_z \leq 1.65$. Furthermore, the number of the structures in the spanwise direction at the point of shedding – approximately 13, though this is not immediately clear given their disorganized nature – is in agreement with the expected results.

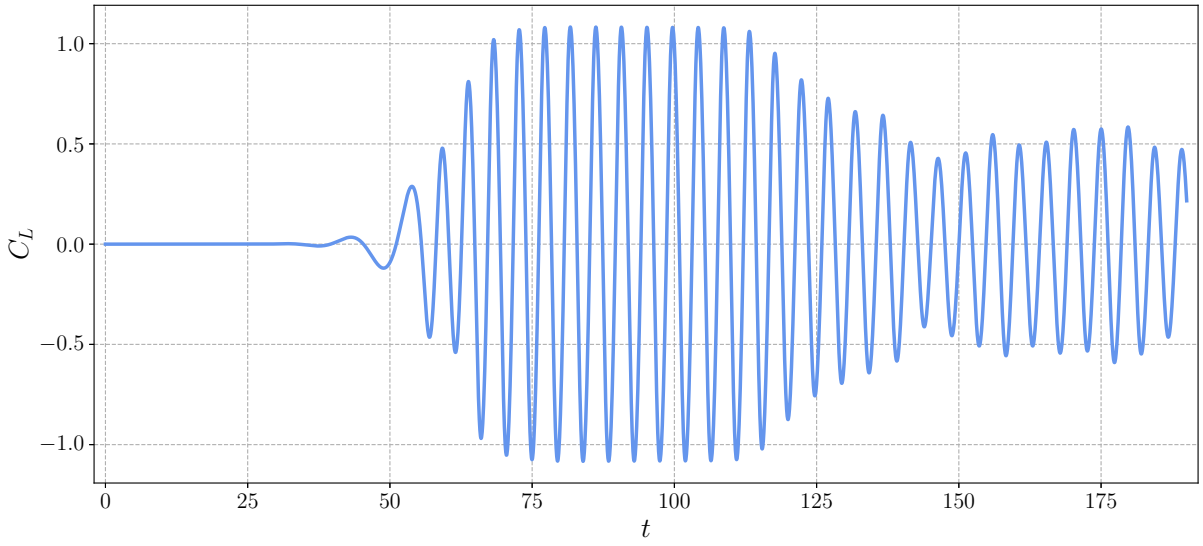


(a)

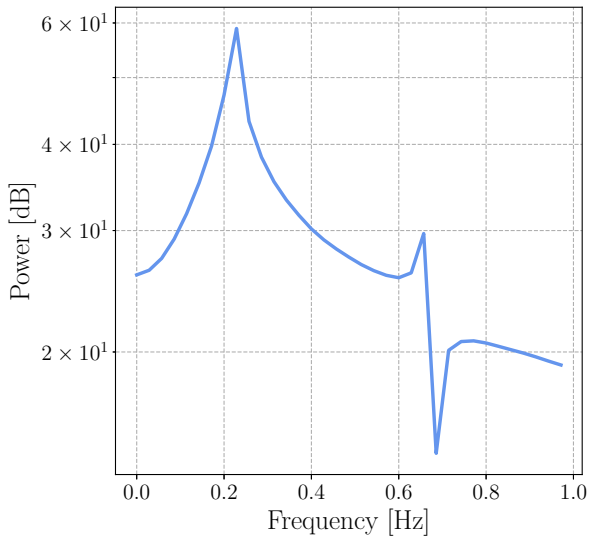


(b)

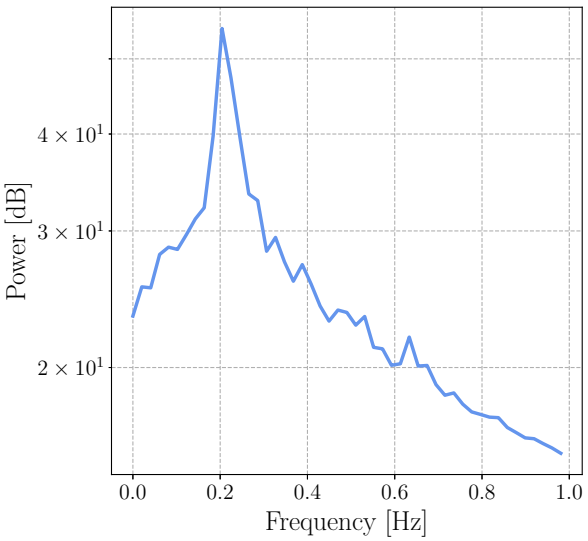
Figure 6.4. Flow past a cylinder: (a) full computational mesh, (b) zoomed-in view near the cylinder.



(a)

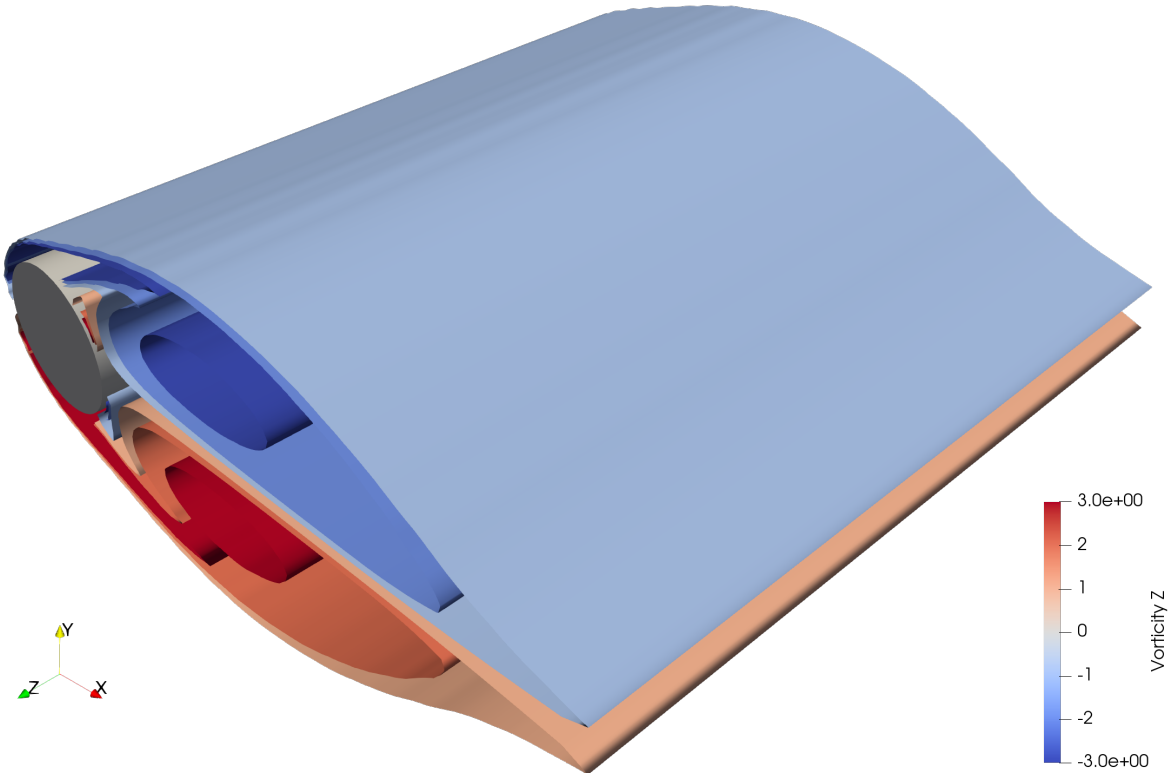


(b)

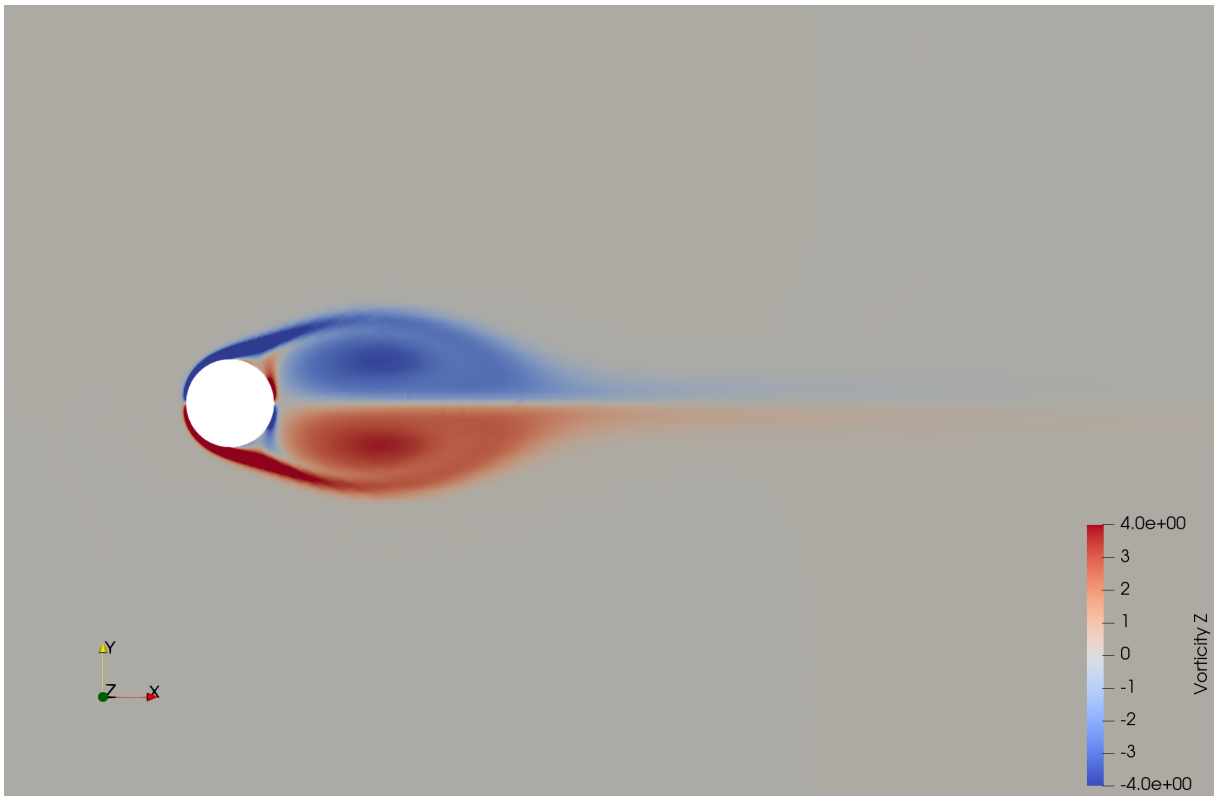


(c)

Figure 6.5. Flow past a cylinder: (a) history of the lift coefficient, (b) power spectrum of the lift force in the interval $65 < t < 110$, (c) power spectrum of the lift force in the fully developed flow.

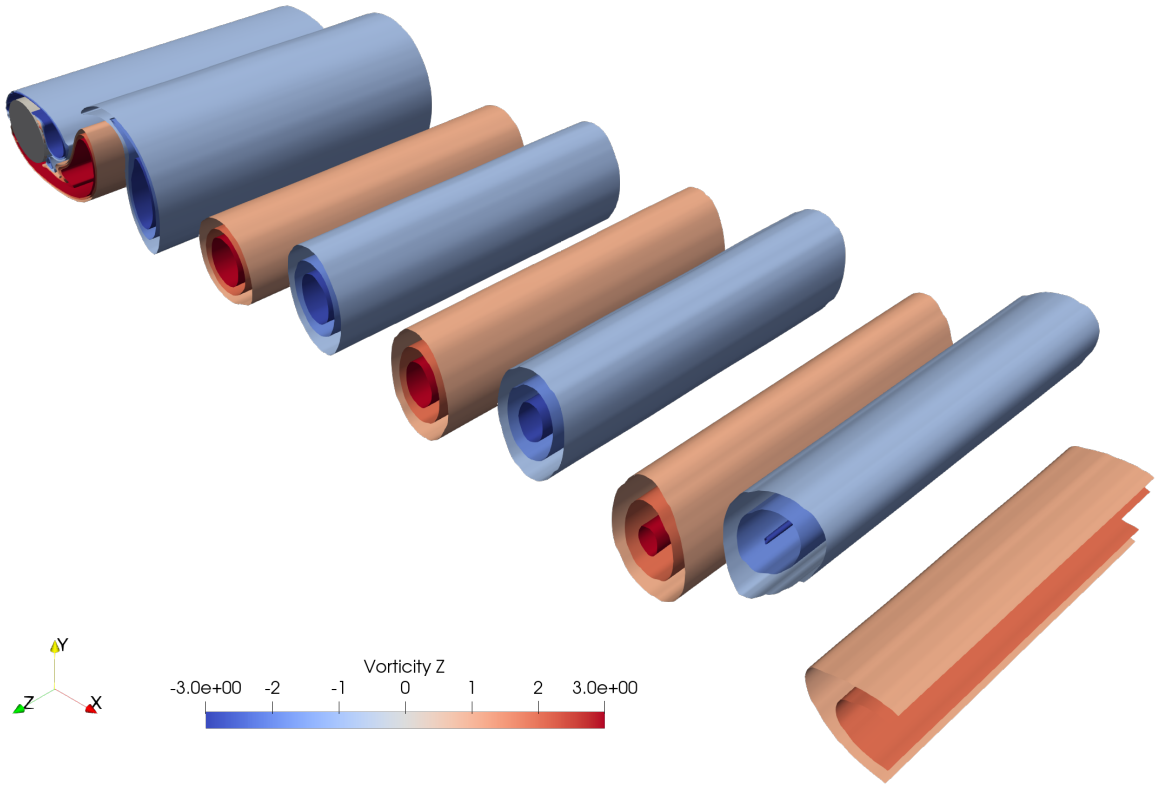


(a)

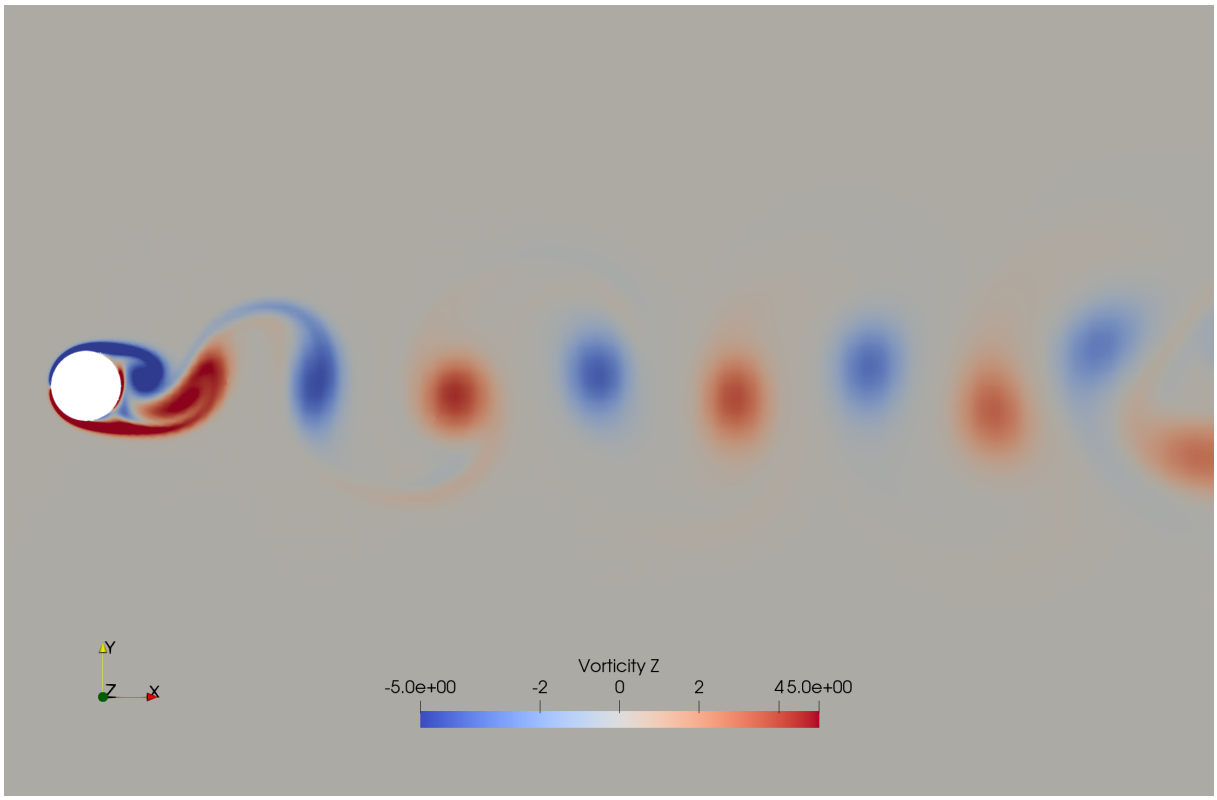


(b)

Figure 6.6. Flow past a cylinder: z component of the vorticity at $t = 15.5$: (a) isocontours, (b) slice taken at $z = 2.5$.

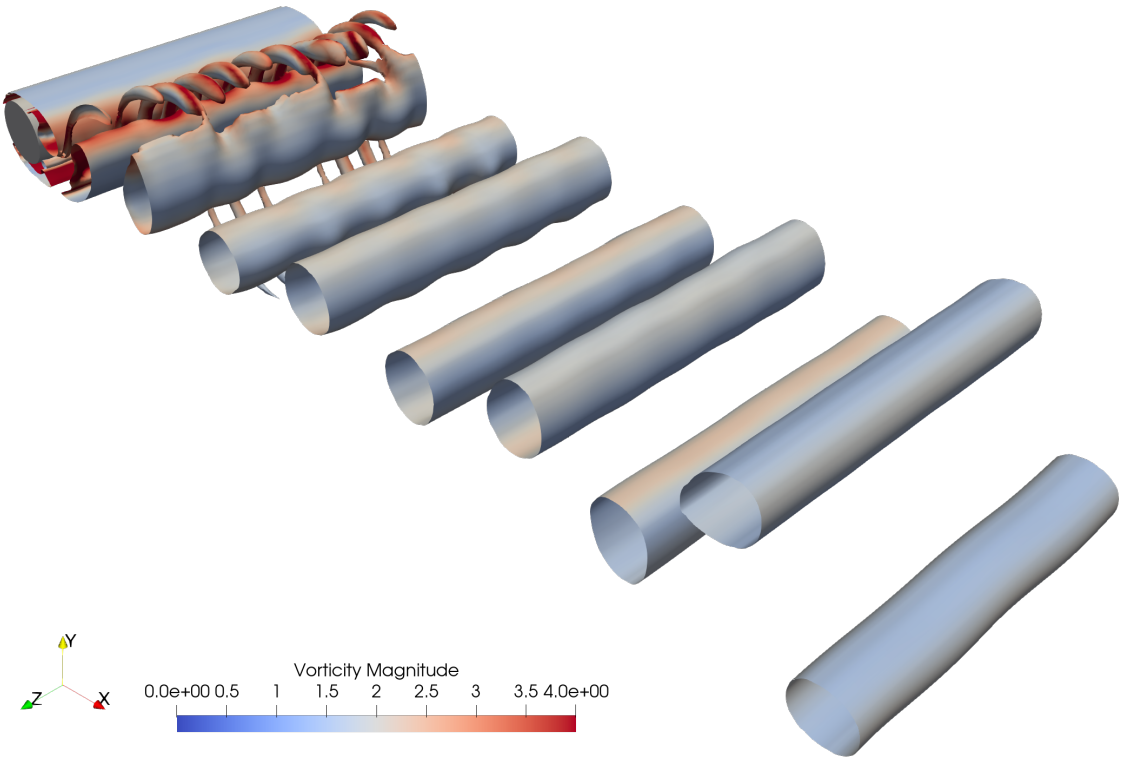


(a)

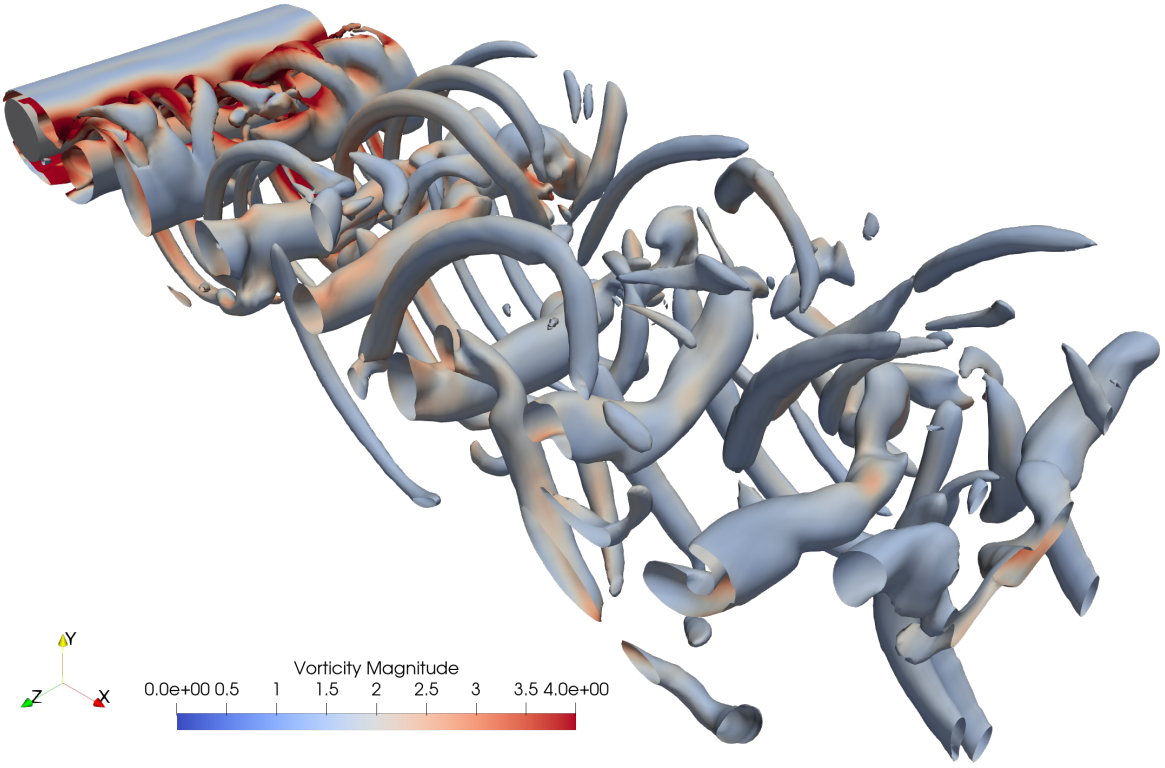


(b)

Figure 6.7. Flow past a cylinder: z component of the vorticity at $t = 90.5$: (a) isocontours, (b) slice taken at $z = 2.5$.

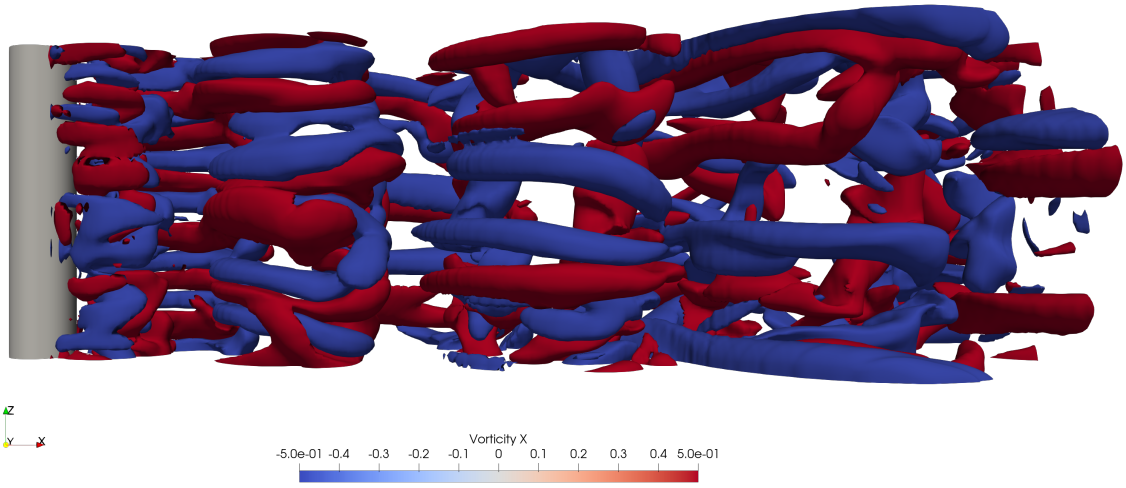


(a)

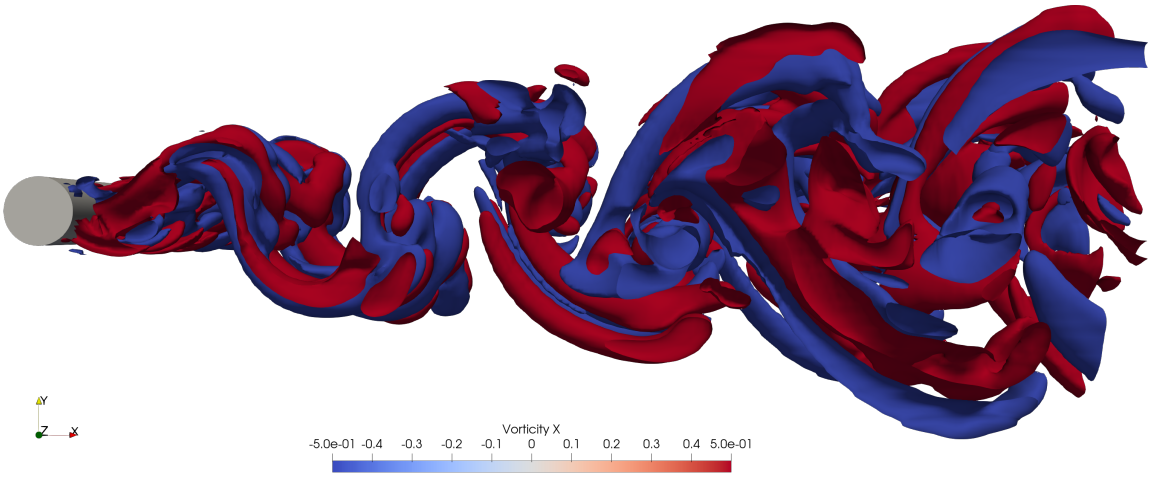


(b)

Figure 6.8. Flow past a cylinder: isocontours of the Q-criterion ($Q = 0.5$) colored by vorticity magnitude at (a) $t = 117.5$, and (b) $t = 155.5$.



(a)



(b)

Figure 6.9. Flow past a cylinder: isocontours of the x component of the vorticity ($\omega_x = \pm 0.5$) in the developed flow at $t = 155.5$. (a) side view, (b) top view.

6.4 Flow in a Rock Fracture

6.4.1 Background and objective

Fracture flows are an area of active research, with important applications in mining engineering and geophysical sciences. The domain in fracture flows consists of the narrow void between two bounding surfaces, and can arise as rocks split due to natural processes, or as the result of human activity. These flows are further characterized by low Reynolds numbers, with orders of magnitude typically in the range of 10^0 to 10^1. The simplest model, which has historically been used to describe such cases, is pressure-driven channel flow between two parallel planes. However, this idealized model is insufficient to capture the effects of wall roughness present in physical fractures [7]. To address this issue, several approaches can be employed, offering varying trade-offs between accuracy and numerical cost [9]. The first of these is the Reynolds lubrication equation, which is a two-dimensional approximation of the Stokes equations, formulated on the assumption that the bounding surfaces are close to parallel. Next, given the low Reynolds number, the Stokes equations can be solved in the three-dimensional domain, discounting inertial effects, but more accurately representing the geometry. Finally, the full Navier-Stokes equations provide the most accurate solution, but require the greatest computational effort. This diversity of flow models presents an opportunity to demonstrate the versatility of the numerical method and solver which is the subject of the present work. To this end, the applicability of the aforementioned approaches to modeling flow in a representative fracture with varying wall roughness will be investigated.

The permeability tensor $\mathbf{K}$ will be used as the point of comparison for the different models. This tensor, possessing the units m^2, describes the flow response of the fracture to a viscosity-normalized external forcing, and is defined as $\mathbf{K} \in \mathbb{R}^{3\times3}$, such that

$$\bar{\mathbf{u}} = \frac{1}{\nu}\mathbf{K}\mathbf{f}, \tag{6.5}$$

where $\mathbf{f}$ is the body force and $\bar{\mathbf{u}}$ is the mean velocity in the domain, defined as

$$\bar{\mathbf{u}} = \frac{1}{|\Omega|}\int_\Omega \mathbf{u}\,\mathrm{d}\mathbf{x}. \tag{6.6}$$

When using linear models, the resulting $\mathbf{K}$ is symmetric positive-definite. Moreover, given the normalization by the viscosity present in (6.5), it does not depend on the Reynolds

number, but solely on the fracture geometry. The reader may find an extensive statistical study of the effect of the fracture roughness on this tensor in [65].

6.4.2 Fracture geometry

The flow domain is given by

$$\Omega = \left\{ \mathbf{x} = (x, y, z) : x, y \in [0, L] \wedge z_1(x, y) \leq z \leq z_2(x, y) \right\}, \tag{6.7}$$

where $z_1, z_2 \in C^0([0, L] \times [0, L])$ parametrize the bounding surfaces. When generating synthetic fractures, it is important to properly define this parametrization, so that the resulting geometries share the statistical properties of real-world fractures [6]. This can be achieved by obtaining z_1 and z_2 as the realizations of the spatially correlated Gaussian fields Z_1 and Z_2, respectively, governed by the Power Spectral Density (PSD) [50]. A robust discussion of this issue can be found in [65]. In the present study, the following simplified PSD was used:

$$P(|\mathbf{f}_s|) = \sigma^2 \begin{cases} \left(\frac{|\mathbf{f}_s|}{f_{\text{roll}}} \right)^{-\alpha} & f > f_{\text{roll}} \\ 1 & \text{otherwise} \end{cases}, \tag{6.8}$$

where P is the power, $\mathbf{f}_s$ is the spatial frequency, σ^2 is the roughness scale, α is the decay ratio, and f_{roll} is the rolling frequency, below which the power is constant. In the present study, the bottom and top surfaces are not correlated, meaning they are independent realizations of the random fields.

The aperture field $h : [0, L] \times [0, L] \to \mathbb{R}$, expressing the clearance in the z direction between the two bounding surfaces, plays an important role in the description of the fracture geometry. Formally, it is defined as

$$h(x, y) = \max(0, z_2(x, y) - z_1(x, y)). \tag{6.9}$$

The mean (reference) aperture h_r is given by

$$h_r = \frac{1}{L^2} \int_0^L \int_0^L h \, \mathrm{d}x \mathrm{d}y. \tag{6.10}$$

Having established this parameter, the Reynolds number in the fracture can be defined by analogy to the channel flow between two plates separated by a distance of h_r, driven

by the body force $\mathbf{f}$. Denoting by $\mathbf{u}_r$ the mean of the resulting parabolic velocity profile, the Reynolds number can be expressed as

$$\mathrm{Re} = \frac{h_r \left|\mathbf{u}_r\right|}{\nu} = \frac{h_r^3 \left|\mathbf{f}\right|}{12\nu^2}. \tag{6.11}$$

Finally, it should be noted that given the definition of the geometry specified above, the z component of the mean velocity must be 0, and the permeability tensor $\mathbf{K}$ can be simplified to a 2 by 2 matrix. Following [65], the following quantities can be defined based on the reduced $\mathbf{K}$:

$$K_a = \frac{1}{2}\left(K_{xx} + K_{yy}\right), \tag{6.12a}$$

$$K_{b1} = \frac{1}{2}\left(K_{yy} - K_{xx}\right), \tag{6.12b}$$

$$K_{b2} = K_{xy}. \tag{6.12c}$$

The isotropic permeability K_a describes the response of the fracture in the direction along which the forcing is applied. Conversely, the quantities K_{b1} and K_{b2}, which will be respectively referred to as the first and second transverse permeabilities in the present text, describe the response in the transverse (anisotropic) direction. In the reference flow between two flat plates the transverse permeabilities are zero, while the isotropic permeability is equal to $\frac{1}{12}h_r^2$. This value, denoted by K_r, will be used as a point of reference.

6.4.3 Reynolds equation

A commonly used approximation, called the Local Cubic Law (LCL) [6], states that the local volumetric flow rate $\mathbf{Q}$ is proportional to the third power of the aperture:

$$\mathbf{Q} = \frac{h^3\left(\mathbf{f} - \nabla p\right)}{12\nu}. \tag{6.13}$$

For $h = const$, the LCL follows immediately from the analytical solution to the channel flow, while for variable h it can be obtained by examining the Reynolds lubrication equation with stationary bounding surfaces. By posing the continuity condition as $\nabla \cdot \mathbf{Q} = 0$ and multiplying out the constants, equation (6.13) leads to the following two-dimensional diffusion equation for the z-averaged pressure:

$$\nabla \cdot \left(h^3 \nabla p\right) = \nabla \cdot \left(h^3 \mathbf{f}\right). \tag{6.14}$$

The equation (6.14) constitutes the first fracture flow model considered in the present study. $\mathbf{Q}$ can be obtained by substituting the calculated pressure gradient back into (6.13). The mean velocity in the domain $\bar{\mathbf{u}} \in \mathbb{R}^2$ can then be computed from

$$\bar{\mathbf{u}} = \frac{1}{L^2 h_r} \int_0^L \int_0^L \mathbf{Q} \, \mathrm{d}x\mathrm{d}y. \tag{6.15}$$

6.4.4 Case description

The fracture geometries were generated in the following manner. First, two independent random surfaces $\hat{z}_1$ and $\hat{z}_2$ conforming to the PSD (6.8) with a mean of 0 were generated using the Rfracture package [64]. Next, the bounding surfaces were obtained using the relation

$$z_1(x, y) = \beta \hat{z}_1(x, y), \tag{6.16a}$$

$$z_2(x, y) = h_r + \beta \hat{z}_2(x, y), \tag{6.16b}$$

for increasing values of the scaling parameter $\beta \in [0, 1]$, in increments of 0.05. The value of β controls the roughness of the fracture: for $\beta = 0$ the fracture surfaces are two parallel plates separated by h_r, while $\beta = 1$ describes the maximum roughness considered. The PSD parameters and geometric dimensions used have been enumerated in Table 6.3. The resulting fracture geometry obtained for $\beta = 1$ has been shown in Figure 6.10. The corresponding aperture field h can be seen in Figure 6.12(a). Note that the minimum aperture is equal to approximately 1% of h_r, meaning that the fracture is open, i.e., z_1 and z_2 never intersect. Furthermore, the surfaces were generated so that the geometry can be considered as periodic, meaning that

$$\forall_{x \in [0,L]} \; z_i(x, 0) = z_i(x, L), \tag{6.17a}$$

$$\forall_{y \in [0,L]} \; z_i(0, y) = z_i(L, y), \tag{6.17b}$$

for $i \in \{1, 2\}$.

For each of the geometries, simulations were performed using each of the three models: the two-dimensional Reynolds equation, and the three-dimensional Stokes and Navier-Stokes equations. The Reynolds number was set to $Re = 20$. In order to avoid duplication of effort, the solution to the Stokes equation was obtained from the first Newton iteration in the nonlinear procedure employed for the solution of the Navier-Stokes equations,

which used the initial guess for the velocity $\mathbf{u}_\star \equiv \mathbf{0}$. For each of the models, two runs were performed: one for $\mathbf{f} = [1,0,0]^T \left[\frac{N}{kg}\right]$, and one for $\mathbf{f} = [0,1,0]^T \left[\frac{N}{kg}\right]$. Periodic boundary conditions were imposed between the corresponding edges parallel to the x and y axes, and faces parallel to the xz and yz planes, for the two- and three-dimensional models, respectively. Additionally, in all cases, the pressure was specified at an arbitrarily chosen point in the domain, in order for the problems to be well-posed. The domain was discretized in the xy plane using a uniform Cartesian grid of 100 elements in either direction. For the three-dimensional meshes, 5 elements were used in the z direction. The node coordinates were obtained by linearly deforming a cuboidal Cartesian grid in the z direction so that the bottom and top surfaces align with z_1 and z_2, respectively, meaning that nodes lying along a line parallel to the z axis were uniformly spaced. The resulting discretization of the bounding surfaces can be seen in Figure 6.10. The approximation order $p = 3$ was used for all runs. Consequently, the number of degrees of freedom equaled to 270,000 and 10,080,000 for the two- and three-dimensional cases, respectively. In all cases, the sum-factorization technique with odd-even decomposition was used with a Jacobi-preconditioned conjugate gradient algebraic solver. The absolute convergence threshold was set to 10^{-8} for the Reynolds equation and 10^{-7} for the three-dimensional models.

When using the Reynolds model, the equation (2.4a) was normalized by a factor of h_r^3. Without this additional scaling, the coefficients of the resulting diffusion operator would differ by 5 orders of magnitude between the first and remaining rows, meaning that the operator would be close to singular. This discrepancy would not only be detrimental to the conditioning of the global operator, but would drastically reduce the contribution of the residual of (2.4a) to the least-squares functional associated with the system (2.4). Tests revealed that even for $\beta = 0$ (i.e. the standard diffusion equation with constant diffusivity), the converged solution was noticeably nonphysical. When the global assembly approach with static condensation was attempted, the results would oscillate in space by multiple orders of magnitude, a consequence of inverting near-singular matrices. The only approach which yielded accurate results was global assembly without static condensation and a direct algebraic solver. The problem described above is not present in the Stokes and Navier-Stokes systems, where the operators do not become singular, even as the viscosity approaches zero. The reader may find a discussion of weighting the contributions of the

individual equations to the least-squares functional, albeit in a slightly different context, in [82].

Parameter	Value	Unit
L	1	m
h_r	40	mm
σ	1	mm
α	4.5	-
f_{roll}	4	$\frac{1}{m}$

Table 6.3. Dimensions and PSD parameters used to generate the fracture geometry.

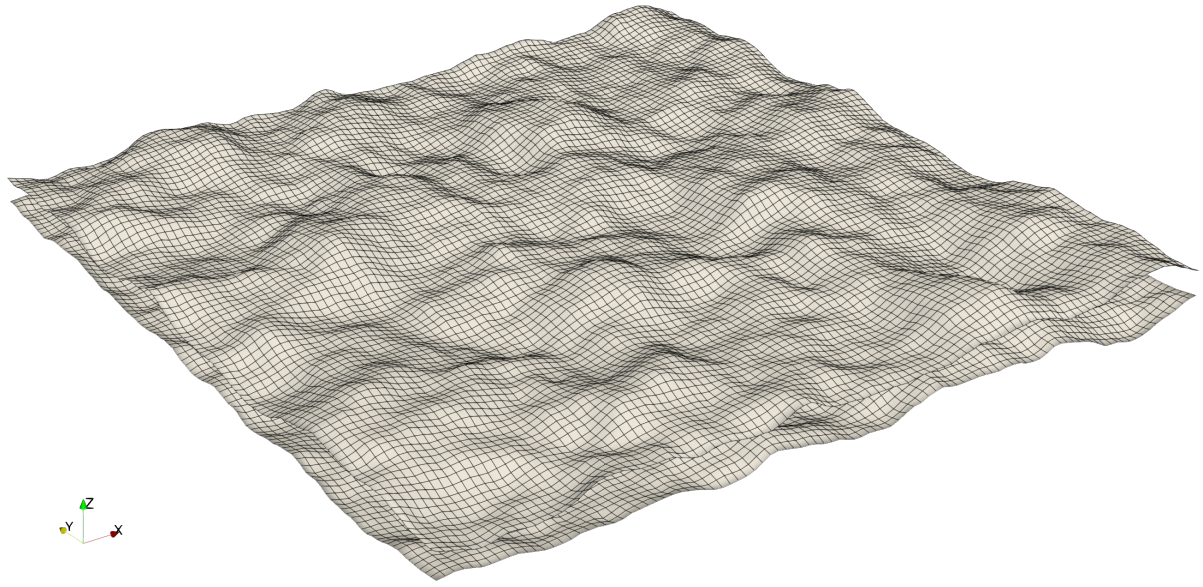


Figure 6.10. Fracture geometry for $\beta = 1$: bounding surfaces z_1 and z_2

6.4.5 Results

For each of the models, at every value of β, the simulations under the two forcings considered provided sufficient data to compute $\mathbf{K}$ using the relation (6.5). The resulting permeabilities have been shown in Figure 6.11. As can be expected, for $\beta = 0$ all models yielded a parabolic velocity profile, in agreement with the analytical solution to channel

flow. However, as the roughness of the fracture increased, its isotropic permeability decreased, and some transverse permeability was introduced. For the maximum roughness considered, the decrease in K_a was equal to 14.6%, 35.7%, and 39% for the Reynolds, Stokes, and Navier-Stokes models, respectively. The respective values of K_{b1} and K_{b2}, normalized by K_r, equaled to 3.35% and 3.36%, 2.55% and 2.35%, and 2.52% and 1.9%. These results clearly indicate that the two-dimensional model is insufficient to accurately capture the characteristics of the fracture, as it severely overpredicts the isotropic permeability and underpredicts the transverse permeability. This finding is in line with other studies, such as [7]. It indicates that when the roughness is significant, the majority of the flow resistance is induced by the tortuosity of the geometry, not merely by the variance in the aperture field. To further substantiate this observation, the streamlines of the flow for the Reynolds and Navier-Stokes models were shown in Figure 6.12. The upstream edge/face was used as the primary source of the stream tracers. In the three-dimensional case, the sources were organized into bundles of three, each bundle aligned and uniformly spaced in the z direction. Qualitatively speaking, similar trends can be observed in the two- and three-dimensional streamlines, e.g., several "bulges" are visible in the central region of the fracture, where the streamlines bend away from areas of low aperture and then reconverge. However, as the three-dimensional rays continue downstream, the bundles clearly lose their orientation in the z direction, and in fact some of them exit the domain nearly aligned along the xy plane. This behavior suggests the presence of helicity in the flow – a fundamentally three-dimensional quality. Moreover, some minor detached vortical structures are present in the flow, as visible in the isocontours of the Q-criterion, shown in Figure 6.13.

Interesting conclusions can also be drawn by comparing the two three-dimensional models considered. Although the results obtained using the Stokes and Navier-Stokes models are much more closely aligned with each other than with those produced by the Reynolds model, the discrepancy is nevertheless noticeable, even at the low Reynolds number $Re = 20$. This suggests that inertial effects play a non-negligible role in the flow dynamics. Furthermore, the permeability tensor computed using the Navier-Stokes equations becomes discernibly non-symmetric for $\beta > 0.8$. For this reason, both the off-diagonal entries of $\mathbf{K}$ obtained from this model have been shown in Figure 6.11(c). Nevertheless, if computational effort poses a major concern, the Stokes model can be

deemed an acceptable choice, particularly if calculating the isotropic permeability is of primary interest. The total number of conjugate gradient iterations required for convergence has been compared in Figure 6.14. As β increases, the savings from employing the linear model become more significant, up to 61% for $\beta = 1$. The Stokes model can therefore be assessed as offering reasonably accurate results at a significantly lower cost.

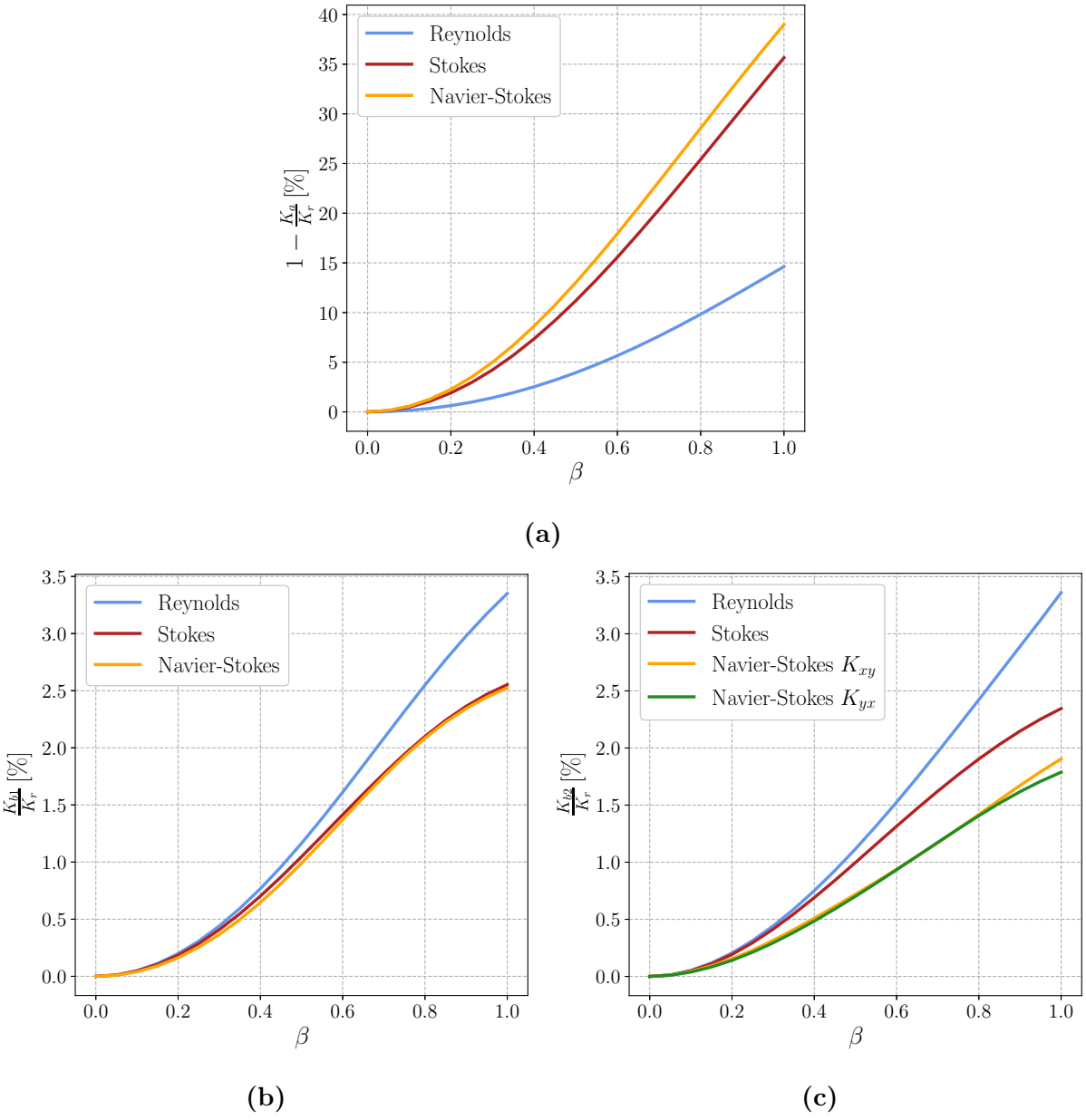
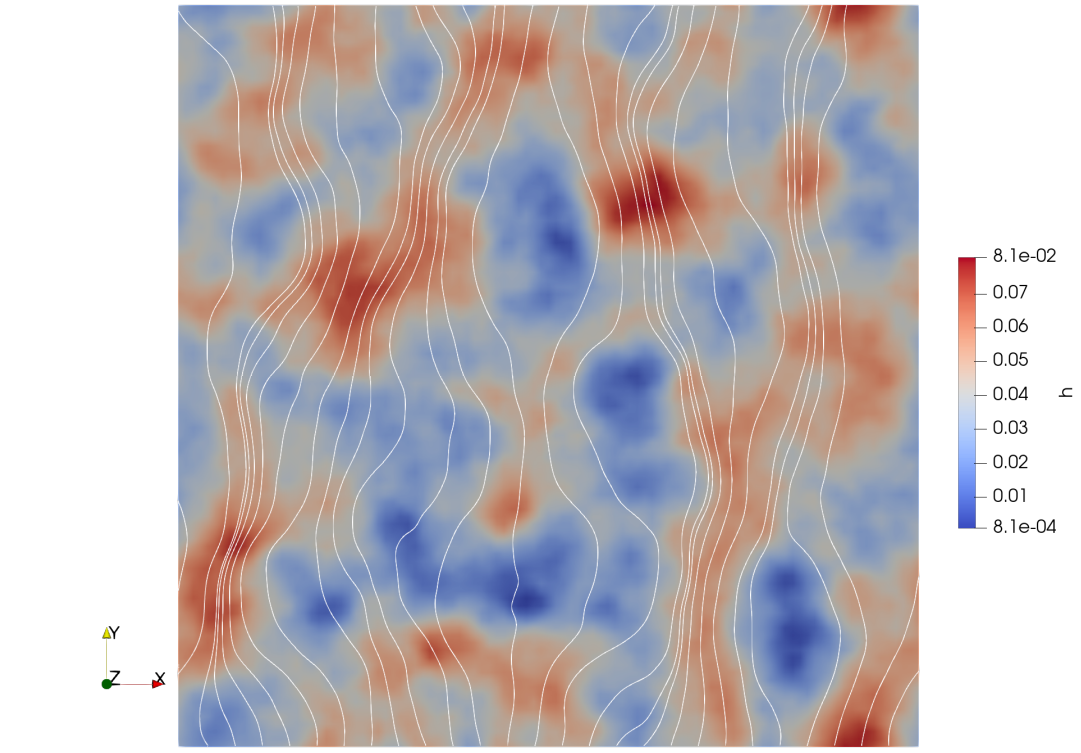
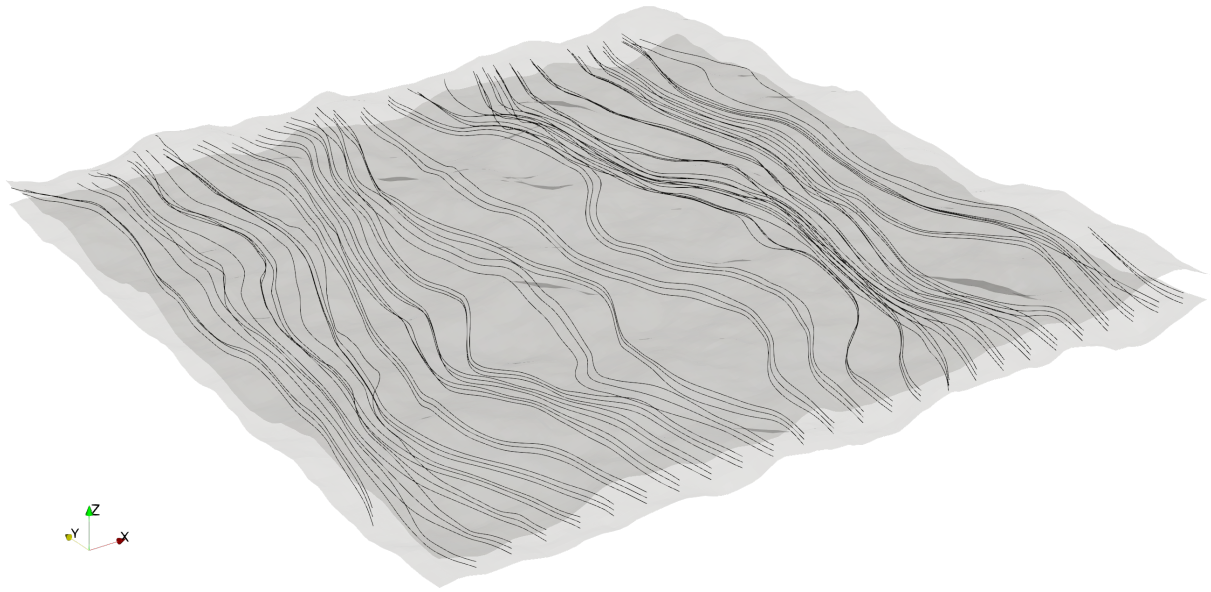


Figure 6.11. Effect of fracture roughness on the permeability tensor for the considered models: (a) reduction in the isotropic permeability K_a compared to the reference solution for flat plates, (b) first transverse permeability K_{b1}, (c) second transverse permeability K_{b2}. Transverse permeability is reported as a percentage of K_r.



(a)



(b)

Figure 6.12. Streamlines of the fracture flow obtained for $\beta = 1$ and $\mathbf{f} = \mathbf{e}_y$: (a) results obtained using the Reynolds equation superimposed on the aperture field h, (b) results obtained using the Navier-Stokes equations.

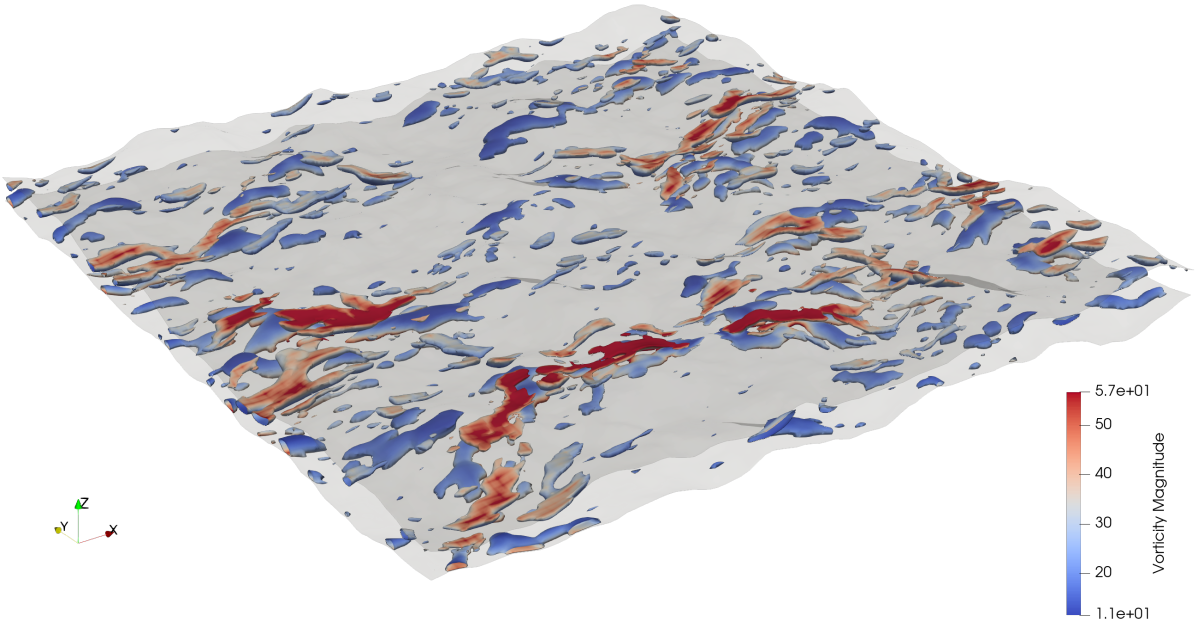


Figure 6.13. Isocontours of the Q-criterion ($Q = 25$) colored by vorticity magnitude. Results obtained using the Navier-Stokes equations for $\mathbf{f} = \mathbf{e}_y$ and $\beta = 1$.

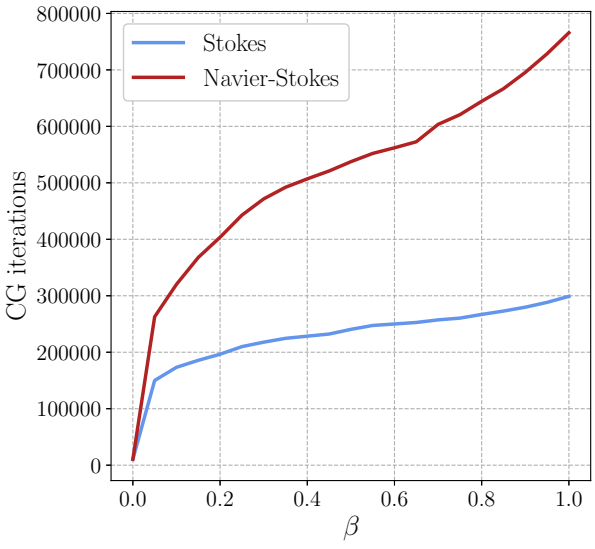


Figure 6.14. Conjugate gradient iterations required for the convergence of the 3D models.

Chapter 7

Conclusions and outlook

In this work, the efficient application of the least-squares spectral/*hp* element method to incompressible flow problems was discussed. The systems of equations governing these physics were reviewed, along with a splitting scheme used to decouple the velocity and pressure components of the solution. Next, the least-squares formulation was presented, and the spectral/*hp* element method described as a basis for the spatial discretization. Subsequently, the different operator evaluation strategies used to apply the action of the discrete least-squares operator were discussed in some detail. The *a priori* operation counts and memory bandwidth costs were presented. Numerical experiments were then undertaken to empirically test the performance of the various approaches. These were conducted using L3STER, the code developed as part of the present work, which was shown to be an efficient and scalable framework for the solution of partial differential equations. Finally, several applications were shown, including convergence studies of the splitting scheme in two and three dimensions, a more computationally demanding DNS of flow past a cylinder, and flow in a geometrically complex fracture.

The most important conclusions of the present work are undoubtedly related to operator evaluation using the sum-factorization technique. As with other formulations of the spectral/*hp* element method, this approach proved key to unlocking the full performance potential of the LSFEM. Its efficiency stems not just from reduced operation counts, but perhaps more importantly from a high computational density, meaning a high ratio of arithmetic operations to memory accesses.

The novel aspects of the present work should be emphasized. The inclusion of open boundary condition in the splitting scheme used herein had not been done previously.

The application of the sum-factorization technique in the context of the least-squares method – which was shown to be indispensable to the efficient evaluation of the resulting operators, particularly in three dimensions – is a new contribution, both in terms of the derivation of the algorithm, and the detailed study of its performance. Finally, L3STER – the solver developed as part of the present work – is a novel numerical tool which can be used by other researchers to conduct physical simulations, possibly even outside the realm of fluid dynamics.

There are several directions in which the research conducted in the present work can be extended. While the computational studies presented herein were performed on a fairly old CPU architecture, the sum-factorization technique enables the efficient utilization of more modern computing hardware, including CPUs with wider vector units (AVX512), and – crucially – GPUs [71]. Extending L3STER to handle a wider breadth of platforms would further involve algorithmic changes to operator evaluation revolving around vectorization of the sum-factorization sweeps across elements. This issue is described in detail in [59]. Other opportunities for future research are related to the time-stepping method used. The "enhanced" convergence rate of the splitting scheme, claimed by its author, was neither demonstrated convincingly nor disproven conclusively, warranting further investigation. The scheme's employment in a fully periodic domain was left as an open problem. Such a setting can be useful, e.g., in the study of homogeneous isotropic turbulence. An altogether different approach, such as the OIFS technique briefly mentioned in the text, could also be explored. Finally, L3STER can be applied in areas different than pure incompressible flow. Multi-physics simulations, such as magnetohydrodynamics (MHD), are a promising direction, given the properties of the underlying numerical method. It is a great hope of the author that the solver developed as part of the present work will prove to be of use to other researchers.

Bibliography

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

[2] M. Antuono. „Tri-periodic fully three-dimensional analytic solutions for the Navier–Stokes equations". In: *Journal of Fluid Mechanics* 890 (2020), A23. DOI: 10.1017/JFM.2020.126.

[3] D. Arndt et al. „The DEAL.II finite element library: Design, features, and insights". In: *Computers and Mathematics with Applications* 81 (2021), pp. 407–422. DOI: 10.1016/j.camwa.2020.02.022.

[4] A. K. Aziz, R. B. Kellogg, and A. B. Stephens. „Least squares methods for elliptic systems". In: *Mathematics of Computation* 44 (1985), pp. 53–70. DOI: 10.1090/S0025-5718-1985-0771030-5.

[5] F. Bertrand and D. Boffi. „First order least-squares formulations for eigenvalue problems". In: *IMA Journal of Numerical Analysis* 42 (2022), pp. 1339–1363. DOI: 10.1093/imanum/drab005.

[6] S. R. Brown. „Simple mathematical model of a rough fracture". In: *Journal of Geophysical Research: Solid Earth* 100 (1995), pp. 5941–5952. DOI: 10.1029/94JB03262.

[7] S. R. Brown, H. W. Stockman, and S. J. Reeves. „Applicability of the Reynolds Equation for modeling fluid flow between rough surfaces". In: *Geophysical Research Letters* 22 (1995), pp. 2537–2540. DOI: 10.1029/95GL02666.

[8] C. Bruneau and P. Fabrie. „Effective Downstream Boundary Conditions for Incompressible Navier-Stokes Equations". In: *International Journal for Numerical Methods in Fluids* 19 (1994), pp. 693–705. DOI: 10.1002/fld.1650190805.

[9] D. J. Brush and N. R. Thomson. „Fluid flow in synthetic rough-walled fractures: Navier-Stokes, Stokes, and local cubic law simulations". In: *Water Resources Research* 39 (2003). DOI: 10.1029/2002WR001346.

[10] „Calculation of impulsively started incompressible viscous flows". In: *International Journal for Numerical Methods in Fluids* 46 (2004), pp. 877–902. DOI: 10.1002/fld.743.

[11] C. D. Cantwell et al. „From h to p Efficiently: Selecting the Optimal Spectral/hp Discretisation in Three Dimensions". In: *Mathematical Modelling of Natural Phenomena* 6 (2011), pp. 84–96. DOI: 10.1051/mmnp/20116304.

[12] C. D. Cantwell et al. „Nektar++: An open-source spectral/hp element framework". In: *Computer Physics Communications* 192 (2015), pp. 205–219. DOI: 10.1016/j.cpc.2015.02.008.

[13] C. Cantwell et al. „From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements". In: *Computers & Fluids* 43 (2011), pp. 23–28. DOI: 10.1016/j.compfluid.2010.08.012.

[14] C. Canuto et al. *Spectral Methods in Fluid Dynamics*. Springer Berlin Heidelberg, 1988. DOI: 10.1007/978-3-642-84108-8.

[15] C. Canuto et al. *Spectral Methods*. Springer Berlin Heidelberg, 2006. DOI: 10.1007/978-3-540-30726-6.

[16] B. Cockburn, G. E. Karniadakis, and C.-W. Shu, eds. *Discontinuous Galerkin Methods*. Vol. 11. Springer Berlin Heidelberg, 2000. DOI: 10.1007/978-3-642-59721-3.

[17] L. Demkowicz. *Computing with hp-Adaptive Finite Elements. Volume 1: One and Two Dimensional Elliptic and Maxwell Problems*. 1st. Chapman and Hall/CRC, 2006. DOI: 10.1201/9781420011685.

[18] L. Demkowicz et al. *Computing with hp-AdaptiveFinite Elements. Volume 2: Three-Dimensional Elliptic and Maxwell Problems with Applications*. Chapman and Hall/CRC, 2007. DOI: 10.1201/9781420011692.

[19] L. F. Demkowicz and J. Gopalakrishnan. „An Overview of the Discontinuous Petrov Galerkin Method". In: Springer, Cham, 2014, pp. 149–180. DOI: 10.1007/978-3-319-01818-8_6.

[20] M. O. Deville, P. F. Fischer, and E. H. Mund. *High-Order Methods for Incompressible Fluid Flow*. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511546792.

[21] S. Dong. „A convective-like energy-stable open boundary condition for simulations of incompressible flows". In: *Journal of Computational Physics* 302 (2015), pp. 300–328. DOI: 10.1016/j.jcp.2015.09.017.

[22] S. Dong. „An outflow boundary condition and algorithm for incompressible two-phase flows with phase field approach". In: *Journal of Computational Physics* (2014). DOI: 10.1016/j.jcp.2014.02.011.

[23] S. Dong, G. Karniadakis, and C. Chryssostomidis. „A robust and accurate outflow boundary condition for incompressible flow simulations on severely-truncated unbounded domains". In: *Journal of Computational Physics* 261 (2014), pp. 83–105. DOI: 10.1016/j.jcp.2013.12.042.

[24] S. Dong and J. Shen. „A Pressure Correction Scheme for Generalized Form of Energy-Stable Open Boundary Conditions for Incompressible Flows". In: *Journal of Computational Physics* 291 (2015), pp. 254–278. DOI: 10.1016/j.jcp.2015.03.012.

[25] M. R. Eslami. *Finite Elements Methods in Mechanics*. Vol. 216. Springer International Publishing, 2014. DOI: 10.1007/978-3-319-08037-6.

[26] C. L. Fefferman. *Existence and Smoothness of the Navier-Stokes Equation*. URL: https://www.claymath.org/wp-content/uploads/2022/06/navierstokes.pdf.

[27] U. Fey, M. König, and H. Eckelmann. „A new Strouhal–Reynolds-number relationship for the circular cylinder in the range $47 < Re < 2 \times 10^5$". In: *Physics of Fluids* 10 (1998), pp. 1547–1549. DOI: 10.1063/1.869675.

[28] P. Fischer et al. „Scalability of high-performance PDE solvers". In: *The International Journal of High Performance Computing Applications* 34 (2020), pp. 562–586. DOI: 10.1177/1094342020915762.

[29] Á. Galvão, M. Gerritsma, and B. D. Maerschalck. „*hp*-Adaptive least squares spectral element method for hyperbolic partial differential equations". In: *Journal of Computational and Applied Mathematics* 215 (2008), pp. 409–418. DOI: 10.1016/j.cam.2006.03.063.

[30] J. Gałecki. *L3STER*. URL: https://github.com/kubagalecki/L3STER.

[31] J. Gałecki and J. Szumbarski. „Adjoint-based optimal control of incompressible flows with convective-like energy-stable open boundary conditions". In: *Computers and Mathematics with Applications* 106 (2022), pp. 40–56. DOI: 10.1016/j.camwa.2021.12.004.

[32] G. N. Gatica. *A Simple Introduction to the Mixed Finite Element Method*. Springer International Publishing, 2014. DOI: 10.1007/978-3-319-03695-3.

[33] S. Gepner and J. Floryan. „Flow dynamics in sinusoidal channels at moderate Reynolds numbers". In: *Journal of Fluid Mechanics* 972 (2023), A22. DOI: 10.1017/jfm.2023.719.

[34] A. Ghizzetti and A. Ossicini. *Quadrature Formulae*. Birkhäuser Basel, 1970. DOI: 10.1007/978-3-0348-5836-6.

[35] Google. *Google Benchmark*. URL: https://github.com/google/benchmark.

[36] D. Gottlieb and S. A. Orszag. *Numerical Analysis of Spectral Methods*. Society for Industrial and Applied Mathematics, 1977. DOI: 10.1137/1.9781611970425.

[37] J. L. Guermond, P. Minev, and J. Shen. „Error Analysis of Pressure-Correction Schemes for the Time-Dependent Stokes Equations with Open Boundary Conditions". In: *SIAM Journal on Numerical Analysis* 43 (2005), pp. 239–258. DOI: 10.1137/040604418.

[38] J. Guermond, P. Minev, and J. 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.

[39] J. Guermond and J. Shen. „A new class of truly consistent splitting schemes for incompressible flows". In: *Journal of Computational Physics* 192 (2003), pp. 262–276. DOI: 10.1016/j.jcp.2003.07.009.

[40] M. D. Gunzburger et al. *Least-Squares Finite Element Methods*. 1st ed. Vol. 166. Springer New York, 2009, pp. 1–56. DOI: 10.1007/b13382.

[41] R. J. Guyan. „Reduction of stiffness and mass matrices". In: *AIAA Journal* 3 (1965), pp. 380–380. DOI: 10.2514/3.2874.

[42] R. D. Henderson. „Details of the drag curve near the onset of vortex shedding". In: *Physics of Fluids* 7 (1995), pp. 2102–2104. DOI: 10.1063/1.868459.

[43] M. Hestenes and E. Stiefel. „Methods of conjugate gradients for solving linear systems". In: *Journal of Research of the National Bureau of Standards* 49 (1952), p. 409. DOI: 10.6028/jres.049.044.

[44] J. J. Heys et al. „Enhanced Mass Conservation in Least-Squares Methods for Navier-Stokes Equations". In: *SIAM Journal on Scientific Computing* 31 (2009), pp. 2303–2321. DOI: 10.1137/080721303.

[45] J. Heys et al. „First-order system least-squares (FOSLS) for modeling blood flow". In: *Medical Engineering & Physics* 28 (2006), pp. 495–503. DOI: 10.1016/j.medengphy.2005.10.002.

[46] J. L. Hintze and R. D. Nelson. „Violin Plots: A Box Plot-Density Trace Synergism". In: *The American Statistician* 52 (1998), pp. 181–184. DOI: 10.1080/00031305.1998.10480559.

[47] J. Hunt. „Lewis Fry Richardson and his Contributions to Mathematics, Meteorology, and Models of Conflict". In: *Annual Review of Fluid Mechanics* 30 (1998), pp. xiii–xxxvi. DOI: 10.1146/annurev.fluid.30.1.0.

[48] International Organization for Standardization. *Programming languages — C++ (ISO Standard No. 14882:2024)*. 2024.

[49] A. Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 2008, pp. 1–459. DOI: 10.1017/CBO9780511995569.

[50] T. D. B. Jacobs, T. Junge, and L. Pastewka. „Quantitative characterization of surface topography using spectral analysis". In: *Surface Topography: Metrology and Properties* 5 (2017), p. 013001. DOI: 10.1088/2051-672X/aa51f8.

[51] H. Jiang and L. Cheng. „Strouhal–Reynolds number relationship for flow past a circular cylinder". In: *Journal of Fluid Mechanics* 832 (2017), pp. 170–188. DOI: 10.1017/jfm.2017.685.

[52] B.-n. Jiang. *The Least-Squares Finite Element Method*. Springer Berlin Heidelberg, 1998. DOI: 10.1007/978-3-662-03740-9.

[53] F. Johansson and M. Mezzarobba. „Fast and Rigorous Arbitrary-Precision Computation of Gauss–Legendre Quadrature Nodes and Weights". In: *SIAM Journal on Scientific Computing* 40 (2018), pp. C726–C747. DOI: 10.1137/18M1170133.

[54] G. Karniadakis and S. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics*. 2nd. Oxford University Press, 2005. DOI: 10.1093/acprof:oso/9780198528692.001.0001.

[55] G. Karypis and V. Kumar. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*. Tech. rep. University of Minnesota, 1997.

[56] R. C. Kirby, M. G. Knepley, and L. R. Scott. *Evaluation of the Action of Finite Element Operators*. Tech. rep. University of Chicago, 2004.

[57] P. Knabner, S. Korotov, and G. Summ. „Conditions for the invertibility of the isoparametric mapping for hexahedral finite elements". In: *Finite Elements in Analysis and Design* 40 (2003), pp. 159–172. DOI: 10.1016/S0168-874X(02)00196-8.

[58] D. A. Kopriva. *Implementing Spectral Methods for Partial Differential Equations*. Springer Netherlands, 2009. DOI: 10.1007/978-90-481-2261-5.

[59] M. Kronbichler and K. Kormann. „Fast Matrix-Free Evaluation of Discontinuous Galerkin Finite Element Operators". In: *ACM Transactions on Mathematical Software* 45 (2019), pp. 1–40. DOI: 10.1145/3325864.

[60] M. Kronbichler and K. Ljungkvist. „Multigrid for Matrix-Free High-Order Finite Element Computations on Graphics Processors". In: *ACM Transactions on Parallel Computing* 6 (2019), pp. 1–32. DOI: 10.1145/3322813.

[61] M. Kronbichler and P.-O. Persson. *Efficient High-Order Discretizations for Computational Fluid Dynamics*. Vol. 602. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-60610-7.

[62] E. Lee and H. Na. „Dual least-squares finite element method with stabilization". In: *Numerical Methods for Partial Differential Equations* 39 (2023), pp. 2975–2997. DOI: 10.1002/num.22996.

[63] J. Liu. „Open and traction boundary conditions for the incompressible Navier–Stokes equations". In: *Journal of Computational Physics* 228 (2009), pp. 7250–7267. DOI: 10.1016/j.jcp.2009.06.021.

[64] Ł. Łaniewski-Wołłk. *The Rfracture Package*. URL: https://github.com/llaniewski/rfracture/.

[65] Ł. Łaniewski-Wołłk and C. Leonardi. „Towards a stochastic model of the permeability of rough fractures". In: *Proceedings of the 22nd Australasian Fluid Mechanics Conference AFMC2020*. 2020. DOI: 10.14264/9d96db0.

[66] Y. Maday, A. T. Patera, and E. M. Rønquist. „An Operator-integration-factor splitting method for time-dependent problems: Application to incompressible fluid flow". In: *Journal of Scientific Computing* 5 (1990), pp. 263–292. DOI: 10.1007/BF01063118.

[67] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*. 2023. URL: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf.

[68] E. Mitsoulis and N. A. Malamataris. „Free (open) boundary condition: some experiences with viscous flow simulations". In: *International Journal for Numerical Methods in Fluids* 68 (2012), pp. 1299–1323. DOI: 10.1002/fld.2608.

[69] S. Mohapatra et al. „Non-conforming least-squares spectral element method for Stokes equations on non-smooth domains". In: *Journal of Computational and Applied Mathematics* 372 (2020), p. 112696. DOI: 10.1016/j.cam.2019.112696.

[70] D. Moxey, R. Amici, and M. Kirby. „Efficient Matrix-Free High-Order Finite Element Evaluation for Simplicial Elements". In: *SIAM Journal on Scientific Computing* 42 (2020), pp. C97–C123. DOI: 10.1137/19M1246523.

[71] „Nek5000/RS performance on advanced GPU architectures". In: *Frontiers in High Performance Computing* 2 (2025), p. 1303358.

[72] M. Nickaeen, A. Ouazzi, and S. Turek. „Newton multigrid least-squares FEM for the V-V-P formulation of the Navier-Stokes equations". In: *Journal of Computational Physics* 256 (2014), pp. 416–427. DOI: 10.1016/j.jcp.2013.09.011.

[73] S. A. Orszag. „Numerical Simulation of Incompressible Flows Within Simple Boundaries. I. Galerkin (Spectral) Representations". In: *Studies in Applied Mathematics* 50 (1971), pp. 293–327. DOI: 10.1002/sapm1971504293.

[74] S. A. Orszag. „Spectral methods for problems in complex geometries". In: *Journal of Computational Physics* 37 (1980), pp. 70–92. DOI: 10.1016/0021-9991(80)90005-4.

[75] A. Ouazzi et al. „Newton-multigrid least-squares fem for s-v-p formulation of the navier-stokes equations". In: *Lecture Notes in Computational Science and Engineering*. Vol. 103. Springer Verlag, 2015, pp. 651–659. DOI: 10.1007/978-3-319-10705-9_64.

[76] K. Park, M. Gerritsma, and M. Fernandino. „C^1 continuous h-adaptive least-squares spectral element method for phase-field models". In: *Computers & Mathematics with Applications* 75 (2018), pp. 1582–1594. DOI: 10.1016/j.camwa.2017.11.026.

[77] K. Park et al. „The least-squares spectral element method for phase-field models for isothermal fluid mixture". In: *Computers & Mathematics with Applications* 74 (2017), pp. 1981–1998. DOI: 10.1016/j.camwa.2017.06.059.

[78] S. Patel et al. *An Operator-Integration-Factor Splitting (OIFS) method for Incompressible Flows in Moving Domains*. Tech. rep. Argonne National Laboratory, 2017.

[79] A. T. Patera. „A spectral element method for fluid dynamics: Laminar flow in a channel expansion". In: *Journal of Computational Physics* 54 (1984), pp. 468–488. DOI: 10.1016/0021-9991(84)90128-1.

[80] G. Payette and J. Reddy. „On the roles of minimization and linearization in least-squares finite element models of nonlinear boundary-value problems". In: *Journal of Computational Physics* 230 (2011), pp. 3589–3613. DOI: 10.1016/j.jcp.2011.02.002.

[81] J. Pontaza. „Least-squares variational principles and the finite element method: theory, formulations, and models for solid and fluid mechanics". In: *Finite Elements in Analysis and Design* 41 (2005), pp. 703–728. DOI: 10.1016/j.finel.2004.09.002.

[82] J. Pontaza. „A least-squares finite element formulation for unsteady incompressible flows with improved velocity–pressure coupling". In: *Journal of Computational Physics* 217 (2006), pp. 563–588. DOI: 10.1016/j.jcp.2006.01.013.

[83] J. Pontaza. „A new consistent splitting scheme for incompressible Navier–Stokes flows: A least-squares spectral element implementation". In: *Journal of Computational Physics* 225 (2007), pp. 1590–1602. DOI: 10.1016/j.jcp.2007.02.009.

[84] J. Pontaza and J. Reddy. „Spectral/hp least-squares finite element formulation for the Navier–Stokes equations". In: *Journal of Computational Physics* 190 (2003), pp. 523–549. DOI: 10.1016/S0021-9991(03)00296-1.

[85] J. Pontaza and J. Reddy. „Least-squares finite element formulations for viscous incompressible and compressible fluid flows". In: *Computer Methods in Applied Mechanics and Engineering* 195 (2006), pp. 2454–2494. DOI: 10.1016/j.cma.2005.05.018.

[86] V. Prabhakar, J. Pontaza, and J. Reddy. „A collocation penalty least-squares finite element formulation for incompressible flows". In: *Computer Methods in Applied Mechanics and Engineering* 197 (2008), pp. 449–463. DOI: 10.1016/j.cma.2007.06.013.

[87] V. Prabhakar and J. Reddy. „Spectral/hp penalty least-squares finite element formulation for the steady incompressible Navier–Stokes equations". In: *Journal of Computational Physics* 215 (2006), pp. 274–297. DOI: 10.1016/j.jcp.2005.10.033.

[88] V. Prabhakar and J. N. Reddy. „Spectral/ hp penalty least-squares finite element formulation for unsteady incompressible flows". In: *International Journal for Numerical Methods in Fluids* 58 (2008), pp. 287–306. DOI: 10.1002/fld.1708.

[89] M. Proot and M. Gerrtisma. „Least-Squares Spectral Elements Applied to the Stokes Problem". In: *Journal of Computational Physics* 181 (2002), pp. 454–477. DOI: 10.1006/jcph.2002.7137.

[90] R. Ranjan and J. N. Reddy. „On multigrid methods for the solution of least-squares finite element models for viscous flows". In: *International Journal of Computational Fluid Dynamics* 26 (2012), pp. 45–65. DOI: 10.1080/10618562.2011.645031.

[91] M. Rieutord. *Fluid Dynamics*. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-09351-2.

[92] E. M. Rønquist and A. T. Patera. „A Legendre spectral element method for the Stefan problem". In: *International Journal for Numerical Methods in Engineering* 24 (1987), pp. 2273–2299. DOI: 10.1002/nme.1620241204.

[93] A. Roshko. *On the Development of Turbulent Wakes from Vortex Streets*. Tech. rep. National Advisory Committee for Aeronautics, 1954.

[94] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003. DOI: 10.1137/1.9780898718003.

[95] R. L. Sani and P. M. Gresho. „Résumé and remarks on the open boundary condition minisymposium". In: *International Journal for Numerical Methods in Fluids* 18 (1994), pp. 983–1008. DOI: 10.1002/fld.1650181006.

[96] J. Scott and M. Tůma. *Algorithms for Sparse Linear Systems*. Springer International Publishing, 2023. DOI: 10.1007/978-3-031-25820-6.

[97] A. Solomonoff. „A fast algorithm for spectral differentiation". In: *Journal of Computational Physics* 98 (1992), pp. 174–177. DOI: 10.1016/0021-9991(92)90182-X.

[98] B. Szabo and I. Babuska. *Finite Element Analysis*. Wiley, 2021, pp. 1–363. DOI: 10.1002/9781119426479.

[99] G. Szegő. *Orthogonal Polynomials*. American Mathematical Society, 1975.

[100] J. Szumbarski et al. „Computations of an unsteady viscous flow in a three-dimensional system of ducts. Part 1: Formulation of mathematical problems and numerical method. Implementation of spectral element method and sample results". In: *Journal of Theoretical and Applied Mechanics* 42 (2004), pp. 21–39.

[101] A. Takhirov and D. Yakoubi. „Pressure Poisson splitting scheme for Navier–Stokes Equations with open boundaries". In: *Mathematics and Computers in Simulation* 240 (2026), pp. 968–981. DOI: 10.1016/j.matcom.2025.08.007.

[102] G. I. Taylor and A. E. Green. „Mechanism of the production of small eddies from large ones". In: *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences* 158 (1937), pp. 499–521. DOI: 10.1098/rspa.1937.0036.

[103] The Trilinos Project Team. *The Trilinos Project Website*. 2025. URL: https://trilinos.github.io.

[104] S. V. Tsynkov. „Numerical solution of problems on unbounded domains. A review". In: *Applied Numerical Mathematics* 27 (1998), pp. 465–532. DOI: 10.1016/S0168-9274(98)00025-7.

[105] B. Turcksin, M. Kronbichler, and W. Bangerth. „WorkStream – A Design Pattern for Multicore-Enabled Finite Element Computations". In: *ACM Transactions on Mathematical Software* 43 (2016), pp. 1–29. DOI: 10.1145/2851488.

[106] V. P. Vallalaa, R. Sadr, and J. N. Reddy. „Higher order spectral/hp finite element models of the Navier–Stokes equations". In: *International Journal of Computational Fluid Dynamics* 28 (2014), pp. 16–30. DOI: 10.1080/10618562.2014.886685.

[107] P. E. Vos, S. J. Sherwin, and R. M. Kirby. „From h to p efficiently: Implementing finite and spectral/hp element methods to achieve optimal performance for low- and high-order discretisations". In: *Journal of Computational Physics* 229 (2010), pp. 5161–5181. DOI: 10.1016/j.jcp.2010.03.031.

[108] W. L. Wendland. *Elliptic systems in the plane*. Pitman, 1979.

[109] F. Zhang, ed. *The Schur Complement and Its Applications*. Vol. 4. Springer-Verlag, 2005. DOI: 10.1007/b105056.

[110] O. C. Zienkiewicz, R. L. Taylor, and D. Fox. *The Finite Element Method for Solid and Structural Mechanics*. Elsevier, 2014, pp. 1–624. DOI: 10.1016/C2009-0-26332-X.

[111] O. C. Zienkiewicz, R. L. Taylor, and P. Nithiarasu. *The Finite Element Method for Fluid Dynamics: Seventh Edition*. 7th ed. Elsevier, 2013, pp. 1–544. DOI: 10.1016/C2009-0-26328-8.

[112] O. Zienkiewicz, R. Taylor, and J. Z. Zhu. *The Finite Element Method: its Basis and Fundamentals*. 7th. Elsevier, 2013. DOI: 10.1016/C2009-0-24909-9.