On the Development and Implementation of High-Order Flux Reconstruction Schemes for Computational Fluid Dynamics

by

Freddie David Witherden

Department of Aeronautics, Imperial College London, South Kensington Campus, London SW7 2AZ, United Kingdom

This thesis is submitted for the degree of
Doctor of Philosophy of Imperial College London

September 2015

Abstract

High-order numerical methods for unstructured grids combine the superior accuracy of high-order spectral or finite difference methods with the geometric flexibility of low-order finite volume or finite element schemes. The Flux Reconstruction (FR) approach unifies various high-order schemes for unstructured grids within a single framework. Additionally, the FR approach exhibits a significant degree of element locality, and is thus able to run efficiently on modern streaming architectures, such as graphics processing units (GPUs). The aforementioned properties of FR mean it offers a promising route to performing affordable, and hence industrially relevant, scale-resolving simulations of hitherto intractable unsteady flows within the vicinity of real-world engineering geometries. In this thesis a formulation of the FR approach that is suitable for solving non-linear advection-diffusion type problems on mixed curvilinear grids is developed. Issues around aliasing are explored in detail and techniques for mitigation outlined. A methodology for identifying symmetric quadrature rules inside of a variety of domains is also presented and used to find several rules that appear to be an improvement over those in literature. This methodology is also used to obtain improved sets of solution points inside of triangular elements. PyFR, an open-source Python based framework for solving the compressible Navier–Stokes equations using the FR approach, is also developed. It is designed to target a range of hardware platforms via use of an in-built domain specific language based on the Mako templating engine. PyFR is able to operate on mixed grids in both two and three dimensions and can target NVIDIA GPUs, AMD GPUs, and Intel CPUs. Results are presented for various benchmark flow problems, single-node performance is discussed, heterogeneous multi-node capabilities are analysed, and scalability is demonstrated on up to 2 000 NVIDIA K20X GPUs for a sustained performance of 1.3 PFLOP/s.

Acknowledgements

I would like to begin by thanking my supervisor Dr Peter Vincent for giving me the opportunity to undertake a PhD in his research group. Despite his busy schedule Peter has always been available to discuss technical matters and to support me wherever possible. He also gave me the freedom to pursue my own research interests and for this I am extremely grateful. I would also like to thank my co-supervisor Prof. Spencer Sherwin and Prof. Paul Kelly from the Department of Computing. I also have greatly enjoyed the company of my colleagues in E256: Jovan, Charles, Jeremy, Abeed, Carla, Leon, Edward, Paola, Ilan, Giorgio, Jingxuan, and Xingsi. They have been a wonderful distraction from work and I will always remember the good times we had together. Finally, I would like to thank my parents for their encouragement and support; this thesis is dedicated to them.

Declaration of Originality

The work hereby presented is based on research carried out by the author at the Department of Aeronautics of Imperial College London, and it is all the author's own work except where otherwise acknowledged. No part of the present work has been submitted elsewhere for another degree or qualification.

Copyright Declaration

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work

Contents

List of Figures 8

List of Tables 11

List of Algorithms 13

List of Publications 14

Nomenclature 17

1 Introduction 19

2 Flux Reconstruction 25
- 2.1 Formulation for Mixed Curvilinear Grids . . . 25
- 2.2 Time Stepping . . . 30
- 2.3 Correction Functions . . . 33
- 2.4 Aliasing . . . 35
- 2.5 Matrix Representation . . . 40
- 2.6 Governing Systems . . . 44

3 Quadrature Rules 47
- 3.1 Basis Polynomials . . . 48
- 3.2 Symmetry Orbits . . . 50
- 3.3 Reference Domains . . . 51
- 3.4 Methodology . . . 57
- 3.5 Implementation . . . 61
- 3.6 Rules . . . 61

4 Solution and Flux Points 65
- 4.1 Requirements . . . 65

4.2 Line Segments . . . 68
4.3 Triangles . . . 69

5 Implementation **77**
5.1 Definition of Operator Matrices . . . 77
5.2 Specification of State Matrices . . . 78
5.3 Matrix Multiplication Kernels . . . 78
5.4 Pointwise Kernels . . . 79
5.5 Distributed Memory Parallelism . . . 83

6 Validation **85**
6.1 Euler Vortex . . . 85
6.2 Couette Flow . . . 86
6.3 Cylinder Flow at $Re = 3\,900$ . . . 92
6.4 Single-Node Performance . . . 101
6.5 Multi-Node Heterogeneous Performance . . . 109
6.6 Scalability . . . 113

7 Conclusion **116**

A Approximate Riemann Solvers **118**
A.1 Rusanov . . . 118

B Boundary Conditions **119**
B.1 Supersonic Inflow . . . 119
B.2 Subsonic Outflow . . . 120
B.3 No-slip Isothermal Wall . . . 120
B.4 Characteristic Riemann Invariant Far-Field . . . 120

Bibliography **122**

Colophon **131**

List of Figures

1.1 Trends in the peak floating point performance and memory bandwidth of Intel processors from 1994–2014. Data courtesy of Jan Treibig. . . 22

2.1 Solution points and flux points for a triangle and quadrangle in physical space. For the top edge of the quadrangle normal vectors have been plotted. Observe how the flux points at the interface between the two elements are co-located. 27
2.2 Three four-point collocation projections of $f(x) \in \mathcal{P}^6$. 35
2.3 Packing methodologies for $N_v = 2$ and $|\mathbf{\Omega}_e| = 9$. 40

3.1 Reference domains in two dimensions. 51
3.2 Reference domains in three dimensions. 54

4.1 Origins of non-unisolvency. 66
4.2 A set points that are not unisolvent. 66
4.3 Initial density profile for the vortex in $\mathbf{\Omega}$. The black box shows the area where the error is calculated at $t = 0$. This box remains centred on the vortex as it progress in the $+y$ direction. 72
4.4 The mesh used for the vortex test case consisting of 800 triangles. . . 72
4.5 Semi-log plots of the Lebesgue constant Λ against truncation error ξ for all point sets. Rules which do not make it to $t = 100$ are indicated by hollow markers. 74
4.6 L^2 error against time for the α-optimised, Λ-optimal, ξ-optimal, σ-optimal, $\langle\sigma\rangle$-optimal, and WS points. 75

5.1 Block-by-panel type matrix multiplications for $\mathbf{C} = \mathbf{AB}$ where $\mathbf{A}$ is a constant operator matrix. 79
5.2 PyFR/Mako source for the `negdivconf` kernel. 80
5.3 Generated OpenMP annotated C code for the `negdivconf` kernel. . . 81

5.4 Generated CUDA code for the `negdivconf` kernel. 82
5.5 Generated OpenCL code for the `negdivconf` kernel. 82
5.6 Flow diagram showing the stages required to compute $\boldsymbol{\nabla} \cdot \mathbf{f}$. Symbols correspond to those of §2.5. For simplicity arguments referencing constant data have been omitted. Memory indirection is indicated by red underlines. Synchronisation points are signified by black horizontal lines. 84

6.1 Initial density profile for the vortex in $\boldsymbol{\Omega}$. 86
6.2 Spatial super accuracy observed for a $\wp = 3$ simulation using DG, SD and HU schemes. Results obtained using PyFR v0.1.0. 87
6.3 Converged steady state energy profile for the two dimensional Couette flow problem. 89
6.4 Unstructured mixed element meshes used for the two dimensional Couette flow problem. 89
6.5 Cutaway of the unstructured hexahedral mesh with 1 004 elements. . . 91
6.6 Cutaways through the two meshes. 94
6.7 Computational effort required for the 119 776 element hexahedral mesh and the mixed mesh with 79 344 prims and 227 298 tetrahedra. 96
6.8 Instantaneous surfaces of iso-density coloured by velocity magnitude. 97
6.9 Averaged wake profiles for Mode-H compared with the numerical results of Lehmkuhl et al. 98
6.10 Averaged wake profiles for Mode-L compared with the numerical results of Lehmkuhl et al. and experimental results of Parnaudeau et al. 98
6.11 Averaged pressure coefficient for Mode-H compared with the numerical results of Ma et al. and Lehmkuhl et al. 99
6.12 Averaged pressure coefficient for Mode-L compared with the numerical results of Lehmkuhl et al. and experimental results of Norberg et al. . 100
6.13 Time-span-average stream-wise velocity profiles for Mode-H compared with the numerical results of Lehmkuhl et al. and Ma et al. . . . 102

6.14 Time-span-average stream-wise velocity profiles for Mode-L compared with the numerical results of Lehmkuhl et al. and experimental results of Parnaudeau et al. 103
6.15 Time-span-average cross-stream velocity profiles for Mode-H compared with the numerical results of Lehmkuhl et al. 104
6.16 Time-span-average cross-stream velocity profiles for Mode-L compared with the numerical results of Lehmkuhl et al. and experimental results of Parnaudeau et al. 105
6.17 Sustained performance of PyFR in GFLOP/s for the various pieces of hardware. The backend used by PyFR is given in parentheses. For the OpenCL backend the initial of the vendor is suffixed. As the NVIDIA OpenCL platform is limited to 4 GiB of memory no results are available for $\wp = 3, 4$. 108
6.18 Sustained performance of PyFR on the multi-node heterogeneous system for each mesh with $\wp = 1, 2, 3, 4$. Lost FLOP/s represent the difference between the achieved FLOP/s and the sum of the E5-2697 (C/OpenMP), K40c (CUDA), and W9100 (OpenCL A) bars in Figure 6.17. 113

List of Tables

1.1 A comparison between various methodologies for spatially discretising partial differential equations. 21

2.1 Butcher tableau. 31
2.2 RK4. 31

3.1 Number of points N_p required for a fully symmetric quadrature rule with positive weights of strength ϕ. Rules with underlines represent improvements over those found in the literature (see text). 64

4.1 Number of rules N_r discovered at each polynomial order $\wp$ and the associated quadrature strengths ϕ. The basis order used for computing the truncation error is indicated by ϕ^+. 70
4.2 L^2 errors at $t = 100$ for the various point sets. 76

5.1 Key functionality of PyFR v1.0.0. 78

6.1 L^2 energy error and orders of accuracy for the Couette flow problem on four mixed meshes. The mesh spacing was approximated as $h \sim N_E^{-1/2}$ where N_E is the total number of elements in the mesh. 90
6.2 L^2 energy errors and orders of accuracy for the Couette flow problem on three extruded hexahedral meshes. On account of the extrusion $h \sim N_E^{-1/2}$ where N_E is the total number of elements in the mesh. . . . 91
6.3 L^2 energy errors and orders of accuracy for the Couette flow problem on three unstructured hexahedral meshes. Mesh spacing was taken as $h \sim N_E^{-1/3}$ where N_E is the total number of elements in the mesh. . . . 92
6.4 Approximate memory requirements of PyFR for the two cylinder meshes. 95
6.5 Comparison of quantitative values with experimental and DNS results. 100

6.6 Baseline attributes of the three hardware platforms. For the NVIDIA Tesla K40c GPU Boost was left disabled and ECC was enabled. The Intel Xeon E5-2697 v2 was paired with four DDR3-1600 DIMMs with Turbo Boost enabled. 107
6.7 Time to evaluate $\boldsymbol{\nabla} \cdot \mathbf{f}$ normalised by the total number of DOFs. 110
6.8 Partition weights for the multi-node heterogeneous simulation. 112
6.9 Weak scalability of PyFR at $\wp = 4$. 114
6.10 Strong scalability of PyFR at $\wp = 4$. 115

List of Algorithms

2.1 PI step-size control algorithm. Descriptions of $f_{\max}$, $f_{\min}$, f_{safe}, α, and β can be found in the text. 33

3.1 Procedure for generating symmetric quadrature rules of strength ϕ with N_p points inside of a domain. 62

List of Publications

Parts of the work presented in this thesis have been disseminated through a number of written publications and oral communications; these are listed below, as of September 2015.

Journal Papers

1. FD Witherden and PE Vincent. An Analysis of Solution Point Coordinates for Flux Reconstruction Schemes on Triangular Elements. *Journal of Scientific Computing*, 61(2):398–423, 2014.
2. FD Witherden, AM Farrington, and PE Vincent. PyFR: An Open Source Framework for Solving Advection-Diffusion Type Problems on Streaming Architectures Using the Flux Reconstruction Approach. *Computer Physics Communications*, 185(11):3028–3040, 2014.
3. FD Witherden and PE Vincent. On the Identification of Symmetric Quadrature Rules for Finite Element Methods. *Computers & Mathematics with Applications*, 69(10):1232–1241, 2015.
4. FD Witherden, BC Vermeire, and PE Vincent. Heterogeneous computing on mixed unstructured grids with PyFR. *Computers & Fluids*, 120:173–186, 2015.
5. PE Vincent, AM Farrington, FD Witherden, and A Jameson. An extended range of stable-symmetric-conservative flux reconstruction correction functions. *Computer Methods in Applied Mechanics and Engineering*, 296:248–272, 2015.

Conference Papers

1. G Mengaldo, D De Grazia, FD Witherden, AM Farrington, PE Vincent, SJ Sherwin, and J Peiro. A Guide to the Implementation of Boundary Conditions in

Compact High-Order Methods for Compressible Aerodynamics. Paper AIAA-2014-2923, *7th AIAA Theoretical Fluid Mechanics Conference*, 16–20 June 2014, Atlanta, Georgia, USA.

2. PE Vincent, FD Witherden, AM Farrington, G Ntemos, BC Vermeire, JS Park, and AS Iyer. PyFR: Next-Generation High-Order Computational Fluid Dynamics on Many-Core Hardware. Paper AIAA-2015-3050, *22nd AIAA Computational Fluid Dynamics Conference*, 22–26 June 2015, Dallas, Texas, USA.
3. BC Vermeire, FD Witherden, and PE Vincent. On the Utility of High-Order Methods for Unstructured Grids: A Comparison Between PyFR and Industry Standard Tools. Paper AIAA-2015-2743, *22nd AIAA Computational Fluid Dynamics Conference*, 22–26 June 2015, Dallas, Texas, USA.

Book Chapters

1. J Enkovaara, M Klemm, and FD Witherden. High Performance Python Offloading. *High Performance Parallelism Pearls Volume 2* pp. 246–269, edited by J Jeffers and J Reinders. Morgan Kaufmann, 2015.

Oral Presentations

1. FD Witherden, AM Farrington, and PE Vincent. PyFR: An Open Source Python Framework for High-Order CFD on Many-Core Platforms. *4th International Congress on Computational Engineering and Sciences*, 19–24 May 2013, Las Vegas, Nevada, USA.
2. FD Witherden and PE Vincent. PyFR: Technical Challenges of Bringing Next Generation Computational Fluid Dynamics to GPU Platforms. *NVIDIA GPU Technology Conference*, 24–27 March 2014, San Jose, California, USA.
3. FD Witherden, BC Vermeire, and PE Vincent. Heterogeneous Computing on Mixed Unstructured Grids with PyFR. *UK Many-Core Developer Conference 2014*, 15 December 2014, Cambridge, UK.

4. FD Witherden and PE Vincent. Heterogeneous Computing with a Homogeneous Codebase. *SIAM CSE 2015*, 14–18 March 2015, Salt Lake City, Utah, USA.
5. PE Vincent, FD Witherden, AM Farrington, G Ntemos, BC Vermeire, JS Park, and AS Iyer. PyFR: Next Generation Computational Fluid Dynamics on GPU Platforms. *NVIDIA GPU Technology Conference*, 17–20 March 2015, San Jose, California, USA.
6. FD Witherden, M Klemm, and PE Vincent. PyFR: Heterogeneous Computing on Mixed Unstructured Grids with Python. *EuroSciPy 2015*, 26–29 August 2015, Cambridge, UK.

Poster Presentations

1. FD Witherden, AM Farrington, and PE Vincent. PyFR: An Open Source Python Framework for High-Order CFD on Many-Core Platforms. *4th International Congress on Computational Engineering and Sciences*, 19–24 May 2013, Las Vegas, Nevada, USA.
2. FD Witherden, AM Farrington, and PE Vincent. PyFR: An Open Source Python Framework for Solving Advection-Diffusion Type Problems on Streaming Architectures. *UK Manycore Developer Conference 2013*, 16–17 December 2013, Oxford, UK.
3. FD Witherden, BC Vermeire, and PE Vincent. PyFR: An Open Source Python Framework for High-Order CFD on Heterogeneous Platforms. *SC14*, 16–21 November 2014, New Orleans, Louisiana, USA.

Nomenclature

Throughout this thesis a convention is adopted in which dummy indices on the right hand side of an expression are summed. For example $C_{ijk} = A_{ijl}B_{ilk} \equiv \sum_l A_{ijl}B_{ilk}$ where the limits are implied from the surrounding context. Unless otherwise stated all indices are assumed to be zero-based.

Functions.

δ_{ij}	Kronecker delta
$\det \mathbf{A}$	Matrix determinant
$\dim \mathbf{A}$	Matrix dimensions
$\deg p$	Polynomial degree

Indices.

e	Element type
n	Element number
α	Field variable number
i, j, k	Summation indices
ρ, σ, ν	Summation indices

Domains.

$\mathbf{\Omega}$	Solution domain
$\mathbf{\Omega}_e$	All elements in $\mathbf{\Omega}$ of type e
$\hat{\mathbf{\Omega}}_e$	A *standard* element of type e
$\partial\hat{\mathbf{\Omega}}_e$	Boundary of $\hat{\mathbf{\Omega}}_e$
$\mathbf{\Omega}_{en}$	Element n of type e in $\mathbf{\Omega}$
$\lvert\mathbf{\Omega}_e\rvert$	Number of elements of type e

Expansions.

$\wp$	Polynomial order
N_D	Number of spatial dimensions
N_V	Number of field variables
$\hat{P}_i$	Normalised Legendre polynomial i
$\hat{P}_i^{(\alpha,\beta)}$	Normalised Jacobi polynomial i
$\ell_{e\rho}$	Nodal basis polynomial ρ for element type e
$\psi_{e\rho}$	Orthonormal basis polynomial ρ for element type e
x, y, z	Physical coordinates
$\tilde{x}, \tilde{y}, \tilde{z}$	Transformed coordinates
$\boldsymbol{\mathcal{M}}_{en}$	Transformed to physical mapping

Adornments and suffixes.

$\tilde{\square}$	A quantity in transformed space
$\hat{\square}$	A vector quantity of unit magnitude
$\square^T$	Transpose
$\square^{(u)}$	A quantity at a solution point
$\square^{(q)}$	A quantity at a solution quadrature point

$\square^{(f)}$	A quantity at a flux point
$\square^{(fq)}$	A quantity at a flux quadrature point
$\square^{(f_\perp)}$	A normal quantity at a flux point

Operators.

$\|\| \cdot \|\|$	L^2 norm
$\|\| \cdot \|\|_\infty$	L^∞ norm
$\mathfrak{C}_\alpha$	Common solution at an interface
$\mathfrak{F}_\alpha$	Common normal flux at an interface
$\mathfrak{B}_\alpha$	Ghost solution at a boundary

Chapter 1

Introduction

There is an increasing desire amongst industrial practitioners of computational fluid dynamics (CFD) to undertake high-fidelity scale-resolving simulations of transient compressible flows within the vicinity of complex geometries. For example, to improve the design of next generation unmanned aerial vehicles (UAVs), there exists a need to perform simulations—at Reynolds numbers 10^4–10^7 and Mach numbers $M \sim 0.1$–1.0—of highly separated flow over deployed spoilers/air-brakes; separated flow within serpentine intake ducts; acoustic loading in weapons bays; and flow over entire UAV configurations at off-design conditions. In order to perform these simulations it is necessary to solve the compressible Navier–Stokes equations. These take the form of a non-linear conservation law.

When solving the Navier–Stokes equations numerically it is customary to independently discretise space and time. Although there exist a variety of spatial discretisations the three most popular are [1]: the finite difference (FD) method in which the governing system is discretised onto a structured grid of points, the finite volume (FV) method in which the computational domain is decomposed into cells and an integral form of the governing system is solved within each cell, and the finite element (FE) method where the computational domain is decomposed into elements inside of which sits a polynomial that is required to satisfy a variational form of the governing system. All of these methods have been used successfully to solve fluid flow problems throughout both industry and academia.

An important consideration when choosing a discretisation is the order of accuracy. This dictates how the error in the solution will respond to a change in the resolution of the grid. Implementations of the above methods are usually first- or second-order accurate in space. A consequence of this—and one that is especially prevalent within

industry CFD software—is a large degree of numerical dissipation. Such schemes therefore encounter significant difficulties when attempting to simulate fundamentally unsteady phenomena [2]. This leads us to high-order methods, the promise of which is increased accuracy with a decreased computational cost.

The order of accuracy of an FD scheme can be readily increased by simply expanding the size of the stencil. For FV methods the procedure is somewhat more involved. The most popular high-order FV type schemes are the essentially non-oscillatory (ENO) of Harten el al. [3] and the weighted ENO (WENO) schemes of Lui et al. [4]. These schemes use an adaptive stencil through an unstructured grid in order to achieve a high-order reconstruction. The adaptive nature of the stencil allows both ENO and WENO schemes to automatically achieve high-order accuracy in the vicinity of shocks and other discontinuities. High-order FE schemes can be constructed by increasing the degree of the polynomial inside of each element. Such schemes are normally termed continuous Galerkin (CG) methods with elements being coupled by requiring that the approximate solution be piecewise continuous between elements. Further details can be found in the books by Karniadakis and Sherwin [5] and Solin et al. [6]. A popular alternative to CG is the discontinuous Galerkin (DG) finite element method, first introduced by Reed and Hill [7] in 1973 for solving the neutron transport equation. In DG the solution is not required to be continuous between elements with coupling between elements instead being achieved through the calculation of common fluxes at interfaces. This is similar to the coupling that occurs between cells in FV schemes.

Beyond CG and DG another more recent class of high-order schemes for unstructured grids are spectral difference (SD) methods. Originally proposed under the moniker 'staggered-grid Chebyshev multidomain methods' by Kopriva and Kolias in 1996 [8] their use in CFD was popularised by Sun et al. [9]. In 2007 Huynh proposed the flux reconstruction (FR) approach [10]; a unifying framework for high-order schemes for unstructured grids that incorporates both the nodal DG schemes of Hesthaven and Warburton [11] and, at least for a linear flux function, any SD scheme. Following on from this, in 2009 Gao and Wang introduced a closely related set of methods which they refer to as lifting collocation penalty (LCP) schemes [12, 13]. Subsequently, in 2013 Yu [14] showed that in one dimension that the LCP schemes are identical to the

Table 1.1. A comparison between various methodologies for spatially discretising partial differential equations. Adapted from Table 1.1 of Hesthaven and Warburton [11].

	FD	FV	ENO	FE	FR
Complex geometries	∅	★	★	★	★
High-order accurate	★	∅	★	★	★
Explicit semi-discrete form	★	★	★	∅	★
Conservation laws	★	★	★	⊠	★
Elliptic problems	★	⊠	⊠	★	⊠

★ = yes ⊠ = yes, with modifications ∅ = no

FR approach. As such several, authors have adopted the moniker 'corrections procedure via reconstruction' (CPR) as a means of referring to both FR and LCP. Furthermore, Allaneau and Jameson [15] have showed that it is possible to cast some FR schemes as a filtered nodal DG schemes. On account of the large degree of numerical interoperability between these schemes they are herein all referred to as 'FR type' schemes.

A comparison of the various schemes can be seen in Table 1.1. Given that the focus of this work is on solving the compressible Navier–Stokes equations—a conservation law—in the vicinity of complex geometries and are interested in schemes that are high-order accurate it can be seen from the table the most promising candidates are ENO/WENO schemes and the FR approach.

Modern hardware. Over the past two decades improvements in the arithmetic capabilities of processors have significantly outpaced advances in random access memory. Algorithms which have traditionally been compute bound—such as dense matrix-vector products—are now limited instead by the bandwidth to/from memory. This is epitomised in Figure 1.1. Whereas the processors of two decades ago had FLOP/s-per-byte of ~0.2 more recent chips have ratios upwards of ~4. This disparity is not limited to just conventional CPUs. Massively parallel accelerators and co-processors

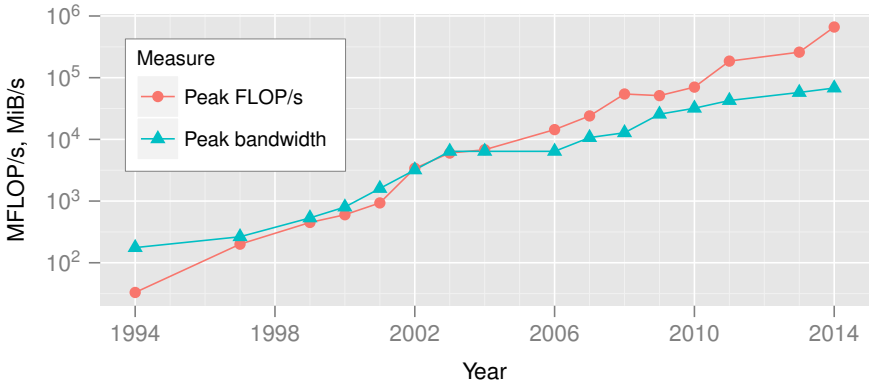


Figure 1.1. Trends in the peak floating point performance and memory bandwidth of Intel processors from 1994–2014. Data courtesy of Jan Treibig.

such as the NVIDIA K20X and Intel Xeon Phi 5110P have ratios of 5.24 and 3.16, respectively.

A concomitant of this disparity is that modern hardware architectures are highly dependent on a combination of high speed caches and/or shared memory to maintain throughput. However, for an algorithm to utilise these efficiently its memory access pattern must exhibit a degree of either spatial or temporal locality. To a first-order approximation the spatial locality of a method is inversely proportional to the amount of memory indirection. On an unstructured grid indirection arises whenever there is coupling between elements. This is potentially a problem for discretisations whose stencil is not compact. Coupling also arises in the context of implicit time stepping schemes. Implementations are therefore very often bound by memory bandwidth. A secondary trend is that the manner in which FLOP/s are realised has also changed. In the early 1990s commodity CPUs were predominantly scalar with a single core of execution. However, in 2015 processors with fourteen or more cores are not uncommon. Moreover, the cores on modern processors almost always contain vector processing units. Vector lengths up to 512-bits, which permit up to eight double precision values to be operated on at once, are rapidly becoming commonplace. It is therefore imperative that compute-bound algorithms are amenable to both multithreading and vectorisation.

A versatile means of accomplishing this is by breaking the computation down into multiple, necessarily independent, streams. By virtue of their independence these streams can be readily divided up between cores and vector lanes.

A corollary of the above discussion is that compute intensive discretisations which can be formulated within the stream processing paradigm are well suited to acceleration on current—and likely future—hardware platforms. Herein lies the primary advantage of the FR approach over competing ENO/WENO type schemes. By working in terms of high-order elements it is possible to achieve a large degree of structured computation. Further, as elements only interact with their nearest neighbour FR schemes can maintain a compact stencil. This is in stark contrast to the large adaptive stencils employed by ENO/WENO methods.

Motivation. Heretofore the majority of the scholarly literature has been concerned with the development of FR schemes that are linearly stable for advection and advection-diffusion problems in a variety of domains. There has been comparatively less work on the non-linear stability of FR schemes and their efficient implementation.

The objective of this work is therefore to help realise the promise of high-order methods for unstructured grids within a real-world setting. A shortcoming of FR, as oft presented in the literature, is that it is generally assumed that the elements are all of the same type, are straight sided, and that the flux is linear. Furthermore, the individual steps of the approach are given in an order which emphasises mathematical clarity over computational efficiency. This can result in implementations which are needlessly restricted in their functionality and perform sub-optimally. Moreover, the majority of treatments forego any discussion of aliasing driven instabilities. However, such instabilities have been found to be a major stumbling block that prevents FR from being used effectively to model unsteady flow phenomena. Additionally, many of the FR codes that have been presented in the literature lack sufficient verification and validation; especially in three dimensions. It is not uncommon for the extension of a piece of work into the third dimension or the efficient implementation thereof to be left as an exercises for the reader.

All of these issues severely inhibit the adoption of these schemes by industry. The

resolution of these issues is hence the primary motivation for this thesis.

Outline. This thesis is organised as follows. In chapter 2 the FR approach is described within the context of solving non-linear advection diffusion type problems on mixed, unstructured, curvilinear grids. Issues around aliasing are discussed and techniques for mitigation outlined. The necessity for numerical quadrature is also assessed. A methodology for determining symmetric quadrature rules is described in chapter 3. The role of solution points, and how they can affect aliasing errors, is discussed in chapter 4. PyFR, an open-source Python based framework for solving the compressible Navier–Stokes equations using the FR approach is described in chapter 5. In chapter 6 various numerical experiments are performed to validate PyFR and showcase its capabilities. Finally, in chapter 7 conclusions are drawn.

Chapter 2

Flux Reconstruction

2.1 Formulation for Mixed Curvilinear Grids

Consider the following advection-diffusion problem inside an arbitrary domain $\mathbf{\Omega}$ in N_D dimensions

$$\frac{\partial u_\alpha}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{f}_\alpha = 0, \tag{2.1}$$

where $0 \leq \alpha < N_V$ is the *field variable* index, $u_\alpha = u_\alpha(\mathbf{x}, t)$ is a conserved quantity, $\mathbf{f}_\alpha = \mathbf{f}_\alpha(u, \boldsymbol{\nabla} u)$ is the flux of this conserved quantity and $\mathbf{x} = x_i$. In defining the flux, u has been taken in its unscripted form to refer to all of the N_V field variables and $\boldsymbol{\nabla} u$ to be an object of length $N_D \times N_V$ consisting of the gradient of each field variable. As a starting point (2.1) is rewritten as a first-order system to give

$$\frac{\partial u_\alpha}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{f}_\alpha(u, \mathbf{q}) = 0, \tag{2.2a}$$

$$\mathbf{q}_\alpha - \boldsymbol{\nabla} u_\alpha = 0, \tag{2.2b}$$

where $\mathbf{q}$ is an auxiliary variable. Here, as with $\boldsymbol{\nabla} u$, $\mathbf{q}$ has been taken in its unsubscripted form to refer to the gradients of all of the field variables.

Take $\mathcal{E}$ to be the set of available element types in N_D dimensions. Examples include quadrilaterals and triangles in two dimensions and hexahedra, prisms, pyramids and tetrahedra in three dimensions. Consider using these various elements types to construct a conformal mesh of the domain such that

$$\mathbf{\Omega} = \bigcup_{e \in \mathcal{E}} \mathbf{\Omega}_e \qquad \text{and} \qquad \mathbf{\Omega}_e = \bigcup_{n=0}^{|\mathbf{\Omega}_e|-1} \mathbf{\Omega}_{en} \qquad \text{and} \qquad \bigcap_{e \in \mathcal{E}} \bigcap_{n=0}^{|\mathbf{\Omega}_e|-1} \mathbf{\Omega}_{en} = \emptyset,$$

where $\mathbf{\Omega}_e$ refers to all of the elements of type e inside of the domain, $|\mathbf{\Omega}_e|$ is the number of elements of this type in the decomposition, and n is an index running over these

elements with $0 \leq n < |\mathbf{\Omega}_e|$. Inside each element $\mathbf{\Omega}_{en}$ it is required that

$$\frac{\partial u_{en\alpha}}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{f}_{en\alpha} = 0, \tag{2.3a}$$

$$\mathbf{q}_{en\alpha} - \boldsymbol{\nabla} u_{en\alpha} = 0. \tag{2.3b}$$

It is convenient, for reasons of both mathematical simplicity and computational efficiency, to work in a transformed space. This is accomplished by introducing, for each element type, a standard element $\hat{\mathbf{\Omega}}_e$ which exists in a transformed space, $\tilde{\mathbf{x}} = \tilde{x}_i$. Next, assume the existence of a mapping function for each element such that

$$x_i = \mathcal{M}_{eni}(\tilde{\mathbf{x}}), \qquad \mathbf{x} = \boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}),$$
$$\tilde{x}_i = \mathcal{M}^{-1}_{eni}(\mathbf{x}), \qquad \tilde{\mathbf{x}} = \boldsymbol{\mathcal{M}}^{-1}_{en}(\mathbf{x}),$$

along with the relevant Jacobian matrices

$$\mathbf{J}_{en} = J_{enij} = \frac{\partial \mathcal{M}_{eni}}{\partial \tilde{x}_j}, \qquad J_{en} = \det \mathbf{J}_{en},$$
$$\mathbf{J}^{-1}_{en} = J^{-1}_{enij} = \frac{\partial \mathcal{M}^{-1}_{eni}}{\partial x_j}, \qquad J^{-1}_{en} = \det \mathbf{J}^{-1}_{en} = \frac{1}{J_{en}}.$$

These definitions provide us with a means of transforming quantities to and from standard element space. Taking the transformed solution, flux, and gradients inside each element to be

$$\tilde{u}_{en\alpha} = \tilde{u}_{en\alpha}(\tilde{\mathbf{x}}, t) = J_{en}(\tilde{\mathbf{x}}) u_{en\alpha}(\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}), t), \tag{2.4a}$$

$$\tilde{\mathbf{f}}_{en\alpha} = \tilde{\mathbf{f}}_{en\alpha}(\tilde{\mathbf{x}}, t) = J_{en}(\tilde{\mathbf{x}}) \mathbf{J}^{-1}_{en}(\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}})) \mathbf{f}_{en\alpha}(\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}), t), \tag{2.4b}$$

$$\tilde{\mathbf{q}}_{en\alpha} = \tilde{\mathbf{q}}_{en\alpha}(\tilde{\mathbf{x}}, t) = \mathbf{J}^{T}_{en}(\tilde{\mathbf{x}}) \mathbf{q}_{en\alpha}(\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}), t), \tag{2.4c}$$

and letting $\tilde{\boldsymbol{\nabla}} = \partial / \partial \tilde{x}_i$, it can be readily verified that

$$\frac{\partial u_{en\alpha}}{\partial t} + J^{-1}_{en} \tilde{\boldsymbol{\nabla}} \cdot \tilde{\mathbf{f}}_{en\alpha} = 0, \tag{2.5a}$$

$$\tilde{\mathbf{q}}_{en\alpha} - \tilde{\boldsymbol{\nabla}} u_{en\alpha} = 0, \tag{2.5b}$$

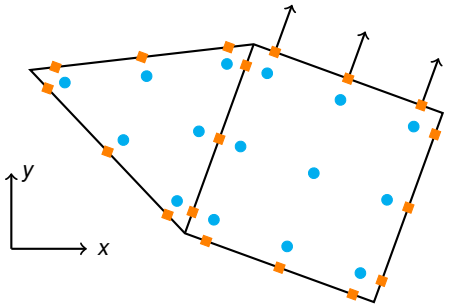


Figure 2.1. Solution points ● and flux points ■ for a triangle and quadrangle in physical space. For the top edge of the quadrangle normal vectors have been plotted. Observe how the flux points at the interface between the two elements are co-located.

as required. Observe here the decision to multiply the first equation through by a factor of J_{en}^{-1}. Doing so has the effect of taking $\tilde{u}_{en} \mapsto u_{en}$ which allows us to work in terms of the physical solution. This is more convenient from a computational standpoint.

The next step in the procedure is to associate a set of solution points with each standard element. For each type $e \in \mathcal{E}$ take $\{\tilde{\mathbf{x}}_{e\rho}^{(u)}\}$ to be the chosen set of points where $0 \leq \rho < N_e^{(u)}(\wp)$. These points can then be used to construct a nodal basis set $\{\ell_{e\rho}^{(u)}(\tilde{\mathbf{x}})\}$ with the property that $\ell_{e\rho}^{(u)}(\tilde{\mathbf{x}}_{e\sigma}^{(u)}) = \delta_{\rho\sigma}$. To obtain such a set first take $\{\psi_{e\sigma}(\tilde{\mathbf{x}})\}$ to be an orthonormal basis which spans a selected order $\wp$ polynomial space defined inside $\hat{\mathbf{\Omega}}_e$. Next, compute the elements of the generalised Vandermonde matrix as $\mathcal{V}_{e\rho\sigma} = \psi_{e\rho}(\tilde{\mathbf{x}}_{e\sigma}^{(u)})$. With these a nodal basis set can be constructed as $\ell_{e\rho}^{(u)}(\tilde{\mathbf{x}}) = \mathcal{V}_{e\rho\sigma}^{-1}\psi_{e\sigma}(\tilde{\mathbf{x}})$. Along with the solution points inside of each element a set of flux points on $\partial\hat{\mathbf{\Omega}}_e$ are also defined. These are denoted for a particular element type as $\{\tilde{\mathbf{x}}_{e\rho}^{(f)}\}$ where $0 \leq \rho < N_e^{(f)}(\wp)$. Let the set of corresponding normalised outward-pointing normal vectors be given by $\{\hat{\tilde{\mathbf{n}}}_{e\rho}^{(f)}\}$. It is critical that each flux point pair along an interface share the same coordinates in physical space. For a pair of flux points $e\rho n$ and $e'\rho' n'$ at a non-periodic interface this can be formalised as $\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}_{e\rho}^{(f)}) = \boldsymbol{\mathcal{M}}_{e'n'}(\tilde{\mathbf{x}}_{e'\rho'}^{(f)})$. A pictorial illustration of this can be seen in Figure 2.1.

The first step in the FR approach is to go from the discontinuous solution at the

solution points to the discontinuous solution at the flux points

$$u^{(f)}_{e\sigma n\alpha} = u^{(u)}_{e\rho n\alpha}\ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}^{(f)}_{e\sigma}), \tag{2.6}$$

where $u^{(u)}_{e\rho n\alpha}$ is an approximate solution of field variable α inside of the nth element of type e at solution point $\tilde{\mathbf{x}}^{(u)}_{e\rho}$. This can then be used to compute a *common solution*

$$\mathfrak{C}_\alpha u^{(f)}_{e\rho n\alpha} = \mathfrak{C}_\alpha u^{(f)}_{\widetilde{e\rho n}\alpha} = \mathfrak{C}_\alpha(u^{(f)}_{e\rho n\alpha}, u^{(f)}_{\widetilde{e\rho n}\alpha}), \tag{2.7}$$

where $\mathfrak{C}_\alpha(u_L, u_R)$ is a scalar function that given two values at a point returns a common value. Here $\widetilde{e\rho n}$ has been taken to be the element type, flux point number and element number of the adjoining point at the interface. Since grids in FR are permitted to be unstructured the relationship between $e\rho n$ and $\widetilde{e\rho n}$ is indirect. This necessitates the use of a lookup table. As the common solution function is permitted to perform upwinding or downwinding of the solution it is in general the case that $\mathfrak{C}_\alpha(u^{(f)}_{e\rho n\alpha}, u^{(f)}_{\widetilde{e\rho n}\alpha}) \neq \mathfrak{C}_\alpha(u^{(f)}_{\widetilde{e\rho n}\alpha}, u^{(f)}_{e\rho n\alpha})$. Hence, it is important that each flux point pair only be visited *once* with the same common solution value assigned to both $\mathfrak{C}_\alpha u^{(f)}_{e\rho n\alpha}$ and $\mathfrak{C}_\alpha u^{(f)}_{\widetilde{e\rho n}\alpha}$.

Further, associated with each flux point is a vector correction function $\mathbf{g}^{(f)}_{e\rho}(\tilde{\mathbf{x}})$ constrained such that

$$\hat{\tilde{\mathbf{n}}}^{(f)}_{e\sigma} \cdot \mathbf{g}^{(f)}_{e\rho}(\tilde{\mathbf{x}}^{(f)}_{e\sigma}) = \delta_{\rho\sigma}, \tag{2.8}$$

with a divergence that sits in the same polynomial space as the solution. Using these fields the solution to (2.5b) can be expressed as

$$\tilde{\mathbf{q}}^{(u)}_{e\sigma n\alpha} = \left[\hat{\tilde{\mathbf{n}}}^{(f)}_{e\rho} \cdot \tilde{\boldsymbol{\nabla}} \cdot \mathbf{g}^{(f)}_{e\rho}(\tilde{\mathbf{x}}) \left\{\mathfrak{C}_\alpha u^{(f)}_{e\rho n\alpha} - u^{(f)}_{e\rho n\alpha}\right\} + u^{(u)}_{e\nu n\alpha}\tilde{\boldsymbol{\nabla}}\ell^{(u)}_{e\nu}(\tilde{\mathbf{x}})\right]_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}^{(u)}_{e\sigma}}, \tag{2.9}$$

where the term inside the curly brackets is the 'jump' at the interface and the final term is an order $\wp - 1$ approximation of the gradient obtained by differentiating the discontinuous solution polynomial. Following the approaches of Kopriva [16] and Sun et al. [9] the physical gradients can now be computed as

$$\mathbf{q}^{(u)}_{e\sigma n\alpha} = \mathbf{J}^{-T\,(u)}_{e\sigma n}\tilde{\mathbf{q}}^{(u)}_{e\sigma n\alpha}, \tag{2.10}$$

$$\mathbf{q}^{(f)}_{e\sigma n\alpha} = \ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}^{(f)}_{e\sigma})\mathbf{q}^{(u)}_{e\rho n\alpha}, \tag{2.11}$$

where $\mathbf{J}_{e\sigma n}^{-T\,(u)} = \mathbf{J}_{en}^{-T}(\tilde{\mathbf{x}}_{e\sigma}^{(u)})$. Having solved the auxiliary equation it is now possible to evaluate the transformed flux

$$\tilde{\mathbf{f}}_{e\rho n\alpha}^{(u)} = J_{e\rho n}^{(u)} \mathbf{J}_{e\rho n}^{-1\,(u)} \mathbf{f}_{\alpha}(u_{e\rho n}^{(u)}, \mathbf{q}_{e\rho n}^{(u)}), \tag{2.12}$$

where $J_{e\rho n}^{(u)} = \det \mathbf{J}_{en}(\tilde{\mathbf{x}}_{e\rho}^{(u)})$. This can be seen to be a collocation projection of the flux. With this the normal transformed flux at each of the flux points can be computed as

$$\tilde{f}_{e\sigma n\alpha}^{(f_\perp)} = \ell_{e\rho}^{(u)}(\tilde{\mathbf{x}}_{e\sigma}^{(f)}) \hat{\mathbf{n}}_{e\sigma}^{(f)} \cdot \tilde{\mathbf{f}}_{e\rho n\alpha}^{(u)}. \tag{2.13}$$

Considering the physical normals at the flux points it is evident that

$$\mathbf{n}_{e\sigma n}^{(f)} = n_{e\sigma n}^{(f)} \hat{\mathbf{n}}_{e\sigma n}^{(f)} = \mathbf{J}_{e\sigma n}^{-T\,(f)} \hat{\hat{\mathbf{n}}}_{e\sigma}^{(f)}, \tag{2.14}$$

which is the outward facing normal vector in physical space where $n_{e\sigma n}^{(f)} > 0$ is defined as the magnitude. As the interfaces between two elements conform it is necessary for $\hat{\mathbf{n}}_{e\sigma n}^{(f)} = -\hat{\mathbf{n}}_{\widetilde{e\sigma n}}^{(f)}$. With these definitions it is now possible to specify an expression for the *common normal flux* at a flux point pair as

$$\mathfrak{F}_{\alpha} f_{e\sigma n\alpha}^{(f_\perp)} = -\mathfrak{F}_{\alpha} f_{\widetilde{e\sigma n}\alpha}^{(f_\perp)} = \mathfrak{F}_{\alpha}(u_{e\sigma n}^{(f)}, u_{\widetilde{e\sigma n}}^{(f)}, \mathbf{q}_{e\sigma n}^{(f)}, \mathbf{q}_{\widetilde{e\sigma n}}^{(f)}, \hat{\mathbf{n}}_{e\sigma n}^{(f)}). \tag{2.15}$$

The relationship $\mathfrak{F}_{\alpha} f_{e\sigma n\alpha}^{(f_\perp)} = -\mathfrak{F}_{\alpha} f_{\widetilde{e\sigma n}\alpha}^{(f_\perp)}$ arises from the desire for the resulting numerical scheme to be conservative; a net outward flux from one element must be balanced by a corresponding inward flux on the adjoining element. It follows that that $\mathfrak{F}_{\alpha}(u_L, u_R, \mathbf{q}_L, \mathbf{q}_R, \hat{\mathbf{n}}_L) = -\mathfrak{F}_{\alpha}(u_R, u_L, \mathbf{q}_R, \mathbf{q}_L, -\hat{\mathbf{n}}_L)$. The common normal fluxes in (2.15) can now be taken into transformed space via

$$\mathfrak{F}_{\alpha} \tilde{f}_{e\sigma n\alpha}^{(f_\perp)} = J_{e\sigma n}^{(f)} n_{e\sigma n}^{(f)} \mathfrak{F}_{\alpha} f_{e\sigma n\alpha}^{(f_\perp)}, \tag{2.16}$$

$$\mathfrak{F}_{\alpha} \tilde{f}_{\widetilde{e\sigma n}\alpha}^{(f_\perp)} = J_{\widetilde{e\sigma n}}^{(f)} n_{\widetilde{e\sigma n}}^{(f)} \mathfrak{F}_{\alpha} f_{\widetilde{e\sigma n}\alpha}^{(f_\perp)}, \tag{2.17}$$

where $J_{e\sigma n}^{(f)} = \det \mathbf{J}_{en}(\tilde{\mathbf{x}}_{e\sigma}^{(f)})$.

It is now possible to compute an approximation for the divergence of the *continuous* flux. The procedure is directly analogous to the one used to calculate the transformed gradient in (2.9)

$$(\tilde{\boldsymbol{\nabla}} \cdot \tilde{\mathbf{f}})_{e\rho n\alpha}^{(u)} = \left[\tilde{\boldsymbol{\nabla}} \cdot \mathbf{g}_{e\sigma}^{(f)}(\tilde{\mathbf{x}}) \left\{ \mathfrak{F}_{\alpha} \tilde{f}_{e\sigma n\alpha}^{(f_\perp)} - \tilde{f}_{e\sigma n\alpha}^{(f_\perp)} \right\} + \tilde{\mathbf{f}}_{e\nu n\alpha}^{(u)} \cdot \tilde{\boldsymbol{\nabla}} \ell_{e\nu}^{(u)}(\tilde{\mathbf{x}}) \right]_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}_{e\rho}^{(u)}}, \tag{2.18}$$

which can then be used to obtain a semi-discretised form of the governing system

$$\frac{\partial u^{(u)}_{e\rho n\alpha}}{\partial t} = -J^{-1\,(u)}_{e\rho n}(\tilde{\boldsymbol{\nabla}} \cdot \tilde{\mathbf{f}})^{(u)}_{e\rho n\alpha}, \tag{2.19}$$

where $J^{-1\,(u)}_{e\rho n} = \det \mathbf{J}^{-1}_{en}(\tilde{\mathbf{x}}^{(u)}_{e\rho}) = 1/J^{(u)}_{e\rho n}$.

2.2 Time Stepping

The semi-discretised form of the governing equation is a system of ordinary differential equations (ODE) in t which can be marched forwards in time using one of a variety of schemes. A popular family of methods that have been applied successfully to advection-diffusion type problems are explicit Runge–Kutta (RK) methods. Given a simple ODE of the form

$$\frac{\mathrm{d}y}{\mathrm{d}t} = g(t, y),$$

an s stage RK scheme is prescribed by

$$g(t + \Delta t) = g(t) + \Delta t b_i k_i, \tag{2.20}$$

$$k_i = g(t + c_i \Delta t, g(t) + a_{ij} k_j), \tag{2.21}$$

where Δt is the desired time step, a_{ij} is an $s \times s$ matrix, b_i is a vector of length s, and $c_i = \sum_j a_{ij}$. These coefficients determine the order of accuracy, q, of a scheme. An RK scheme is explicit if a_{ij} is a strictly lower triangular matrix. In the literature the coefficients of an RK scheme are usually presented in the form of a Butcher tableau as shown in Table 2.1. The coefficients for the classic fourth order four stage "RK4" scheme can be seen in Table 2.2.

Low storage RK schemes. One of the main applications of explicit RK schemes is in solving non-stiff ODEs. Here the primary criterion used in evaluating scheme are its order of accuracy and its stage count. The sweet spot in terms of computational work versus accuracy appears to lie between $q = 5$ and $q = 8$ [17, 18]. However, these ODEs tend to be quite different to those which arise when discretising advection-diffusion type

Table 2.1. Butcher tableau.

c_1	a_{11}	a_{12}	$\dots$	a_{1s}
c_2	a_{21}	a_{22}	$\dots$	a_{2s}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	a_{s1}	a_{s2}	$\dots$	a_{ss}
	b_1	b_2	$\dots$	b_s

Table 2.2. RK4.

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

problems. When solving these ODEs the time step is more often than not restricted by stability requirements as opposed to those of accuracy—the system exhibits a degree of stiffness. Moreover, as there is an ODE associated with each spatial degree of freedom the system can also become extremely large. A consequence of this is that retaining the various intermediate k_i stages in memory can become prohibitively expensive.

For a generic explicit RK scheme it is necessary to allocate storage, termed registers in the literature, for $y(t)$ and each of the s intermediate stages for a total register count of $s+1$. By exploiting the structure of the scheme it is often possible to reduce the register count somewhat. For example, assuming it is possible to evaluate $g(t, y)$ in-place, the RK4 scheme of Table 2.2 can be implemented with just three registers of storage as opposed to five. There exists a significant body of work related to the derivation of low-storage RK schemes [19, 20]. With care it is possible to obtain schemes that require just two registers of storage. Of the schemes the fourth order five stage RK45[2R+] method of Kennedy et al. [20] is notable for its particularly large stability region.

Step size control. In comparison to linear multistep methods each RK time step depends only on the solution at t. It is therefore trivial to change the Δt between steps. Hence, given an approximation of the truncation error $\xi(t + \Delta t)$ it is possible to adapt the step size to both ensure stability and bound the local temporal error. Such control can be used to automatically find the maximum stable step size—eliminating the need for manual bisection.

The most common means of obtaining an approximation to the truncation error is

through an alternative set of b_i coefficients $b_i^\star$ that give a $q-1$ order approximation of the solution. Using this the truncation error can be approximated as $\xi(t+\Delta t) \approx \Delta t(b_i - b_i^\star)k_i$. To be meaningful it is first necessary to normalise the error with respect to a predefined tolerance. Following Hairer et al. [17] an error can be defined as

$$\sigma_{\text{curr}} = \frac{\xi(t+\Delta t)}{\tau_a + \tau_r \max(|y(t)|, |y(t+\Delta t)|)}, \tag{2.22}$$

where τ_a is an absolute error tolerance, and τ_r is a relative error tolerance. When marching a system of equations this expression should be evaluated pointwise for each equation in the system and the root mean square taken. A step should be rejected and retaken with a smaller Δt if $\sigma_{\text{curr}} > 1$. Otherwise the step should be accepted. The objective is to control Δt such that the error incurred during the next step, σ_{next}, is approximately unity. Assuming that the solution is sufficiently smooth it is known that modifying Δt by a factor of f will result in $\sigma_{\text{next}} \approx f^q \sigma_{\text{curr}}$. Hence, to keep the error around unity the adjustment factor should be chosen to be $f \approx \sigma_{\text{curr}}^{-1/q}$. This is known as an "I" type controller. For reasons of computational efficiency it is desirable to minimise the number of rejected time steps. The incidence of such steps can be reduced by firstly introducing a safety factor, $f_{\text{safe}} \approx 0.8$, and secondly by restricting the overall adjustment such that $f_{\min} \leq f_{\text{safe}} f \leq f_{\max}$ where $f_{\min} \approx 0.3$ and $f_{\max} \approx 2.5$.

One problem with I type controllers is that they are prone to spurious oscillations. This issue can be avoided through the use of a "PI" type controller which uses the values of σ from both the current and the previous time steps in order to update Δt. In a PI controller the adjustment factor is calculated as [21, 22]

$$f \approx \sigma_{\text{curr}}^{-\alpha/q} \sigma_{\text{prev}}^{\beta/q}, \tag{2.23}$$

where $\alpha \approx 0.4$ and $\beta \approx 0.7$. Complete pseudocode for a PI controller can be seen in Algorithm 2.1.

The utility of step size control in combination with its ease of implementation has made it ubiquitous within the ODE community. As a result the majority of RK schemes tabulate in literature come with embedded pairs—including the low storage schemes. To evaluate the error it is necessary to have access to both the previous solution $y(t)$

Algorithm 2.1. PI step-size control algorithm. Descriptions of $f_{\max}$, $f_{\min}$, f_{safe}, α, and β can be found in the text.

procedure IntegrateWithPIControl($\Delta t_{\text{init}}, \Delta t_{\min}, t_{\text{end}}$)
  $t \leftarrow 0, f \leftarrow 1, \sigma_{\text{prev}} \leftarrow 0, \Delta t \leftarrow \Delta t_{\text{init}}$
  while $t < t_{\text{end}}$ **do**
    $\Delta t \leftarrow f \Delta t$ ▷ Adjust step size
    $\Delta t \leftarrow \max(\min(t - t_{\text{end}}, \Delta t), \Delta t_{\min})$
    $\sigma_{\text{curr}} \leftarrow \text{Step}(t, \Delta t, \ldots)$ ▷ Take step and compute error
    $f \leftarrow f_{\text{safe}} \sigma_{\text{curr}}^{-\alpha/q} \sigma_{\text{prev}}^{\beta/q}$ ▷ Compute new step adjustment factor
    $f \leftarrow \min(f_{\max}, \max(f_{\min}, f))$ ▷ Ensure $f_{\min} \leq f \leq f_{\max}$
    if $\sigma_{\text{curr}} \leq 1$ **then** ▷ Accept step
      $t \leftarrow t + \Delta t$
      $\sigma_{prev} \leftarrow \sigma_{curr}$
    else if $\Delta t = \Delta t_{\min}$ **then** ▷ Minimum size step rejected
      Abort()
    end if
  end while
end procedure

and the error terms $\xi(t + \Delta t)$. For low storage schemes, which usually operate in-place overwriting $y(t)$ with $g(t, y)$, this requires that two extra registers be allocated.

2.3 Correction Functions

The nature of an FR scheme, including its dispersion and dissipation characteristics [23], its associated Courant-Friedrichs-Lewy (CFL) limit [10, 23], and its fundamental stability [24], are largely determined by the form of the vector correction field. Building on the work of Huynh [10] and Jameson [25], Vincent et al. [24] identified a one

parameter family of correction functions that are provably stable for one dimensional linear advection problems. These are commonly referred to as the Vincent-Castonguay-Jameson-Huynh (VCJH) correction functions. The stability of the VCJH functions was subsequently extended by Castonguay et al. [26] to linear advection-diffusion problems. The FR approach can be extended to quadrilateral and hexahedral elements through a tensor product construction [10]. However, beyond the case of recovering DG, it is an open question if the resulting schemes are linearly stable or not. Further work by Castonguay et al. [27] and Williams et al. [28] has led to the identification of VCJH like schemes inside of triangular elements. These schemes are observed to be distinct from those identified by Huynh [29] in his extension of FR onto triangular elements. Using a similar procedure to Castonguay et al. Williams and Jameson [30] were able to identify a family of VCJH-like schemes inside of tetrahedra. More recently, Vincent et al. [31] developed a procedure for obtaining an extended range of energy stable one dimensional schemes. These schemes are found to be a super-set of the existing one-parameter VCJH schemes.

Here a methodology is presented for obtaining the correction function corresponding to a nodal DG scheme inside of an arbitrary domain. When considering the correction function associated with a flux point $\mathbf{g}_{e\rho}^{(f)}(\tilde{\mathbf{x}})$ it is often more convenient to use a face-local numbering scheme in which $\rho \leftrightarrow (ij)$ where i denotes the face number and j the local index on this face. Let $\{\tilde{\mathbf{\Gamma}}_{ei}\}$ refer to faces of the reference element $\hat{\mathbf{\Omega}}_e$. With these the divergences of the DG correction functions can be expressed as [30, 32]

$$\nabla \cdot \mathbf{g}_{e(ij)}^{(f)}(\tilde{\mathbf{x}}) = \psi_{ek}(\tilde{\mathbf{x}}) \int_{\tilde{\mathbf{\Gamma}}_{ei}} \hat{\mathbf{n}} \cdot \mathbf{g}_{e(ij)}^{(f)}(\tilde{\mathbf{s}}) \psi_{ek}(\tilde{\mathbf{s}}) \, \mathrm{d}\tilde{\mathbf{s}} \tag{2.24}$$

$$= \psi_{ek}(\tilde{\mathbf{x}}) \int_{\tilde{\mathbf{\Gamma}}_{ei}} \ell_{eij}(\tilde{\mathbf{s}}) \psi_{ek}(\tilde{\mathbf{s}}) \, \mathrm{d}\tilde{\mathbf{s}}, \tag{2.25}$$

where $\hat{\mathbf{n}}$ is the outward pointing unit normal vector, and $\ell_{eij}(\tilde{\mathbf{s}})$ is the nodal basis function associated with point j on face i of the reference element e. In the second step the fact that (2.8) fixes $\hat{\mathbf{n}} \cdot \mathbf{g}_{e(ij)}^{(f)}(\tilde{\mathbf{s}})$ at each of the flux points on the face has been utilised to enable it to be substituted for an equivalent nodal basis function. Heretofore this formulation has only been employed for simplex elements—triangles and tetrahedra—however it is valid for any element type.

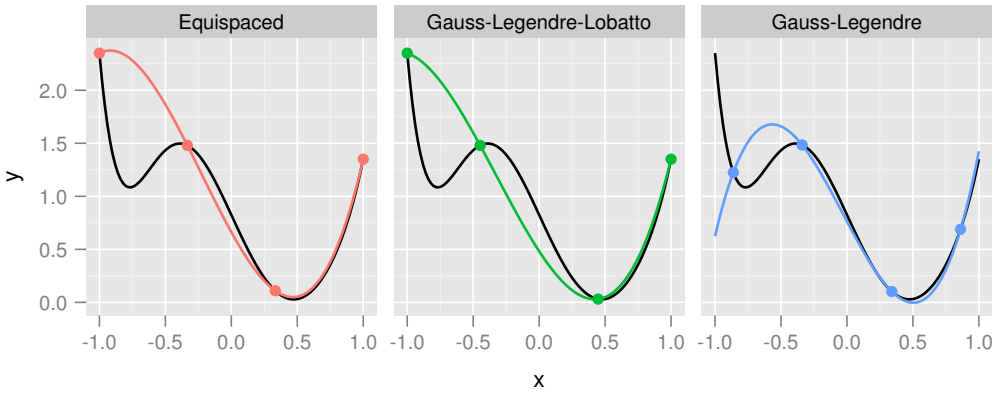


Figure 2.2. Three four-point collocation projections of $f(x) \in \mathcal{P}^6$.

2.4 Aliasing

In the FR approach it is necessary to obtain a suitable approximation of the transformed flux at each of the solution points. The most direct means of accomplishing this is to simply evaluate the transformed flux function at each of the solution points as per (2.12). When $\mathbf{f}$ is non-linear or the grid is curved the transformed flux sits in a different—perhaps even non-polynomial—space to the solution. A consequence of this is that within (2.12) there is an implicit *collocation projection*.

Numerical experiments in one dimension. To investigate the effects of this projection consider introducing polynomial $f(x) \in \mathcal{P}^6$ defined in the range $x \in [-1, 1]$. A collocation projection of $f(x)$ into the space $\mathcal{P}^3$ can be performed by sampling $f(x)$ at four distinct abscissa. The polynomials resulting from three such samplings at equispaced, Gauss-Legendre-Lobatto, and Gauss-Legendre points can be seen in Figure 2.2. In the figure the three polynomials can be seen to have distinct forms. Defining the least squares L^2 error as

$$\sigma^2 = \int_{-1}^{1} \Big[p(x) - f(x)\Big]^2 \,\mathrm{d}x = \int_{-1}^{1} p^2(x) - 2p(x)f(x) + f^2(x) \,\mathrm{d}x, \tag{2.26}$$

and evaluating this expression for the aforementioned three polynomials results in errors of 0.68, 0.60, and 0.40, respectively. These differences in the L^2 error highlight the importance of the choice of solution points when using the FR approach. Having quantified the error it is now possible to minimise it. Expanding $p(x)$ as

$$p(x) = \gamma_i \Upsilon_i(x),$$

where $\{\gamma_i\}$ are a set of expansion coefficients and $\{\Upsilon_i\}$ are a set of basis functions that span $\mathcal{P}^3$. Differentiating (2.26) with respect to γ_k and equating this to zero it is found that

$$\begin{aligned} \partial_{\gamma_j}\sigma^2 &= \int_{-1}^{1} 2p(x)\partial_{\gamma_j}p(x) - 2\partial_{\gamma_j}p(x)f(x)\,\mathrm{d}x \\ &= \int_{-1}^{1} 2\gamma_i\Upsilon_i(x)\Upsilon_j(x) - 2\Upsilon_j(x)f(x)\,\mathrm{d}x \\ &= \gamma_i \int_{-1}^{1} \Upsilon_i(x)\Upsilon_j(x)\,\mathrm{d}x - \int_{-1}^{1} \Upsilon_j(x)f(x)\,\mathrm{d}x \\ &= 0, \end{aligned} \tag{2.27}$$

where in the second step the constant factor of two has been dropped. This can be seen to be a linear system of the form $\mathbf{Ax} = \mathbf{b}$ which are, in general, expensive to solve. However should the basis functions satisfy an orthonormality condition such that

$$\int_{-1}^{1} \Upsilon_i(x)\Upsilon_j(x)\,\mathrm{d}x = \delta_{ij},$$

then (2.27) reduces to

$$\gamma_j = \int_{-1}^{1} \Upsilon_j(x)f(x)\,\mathrm{d}x,$$

which is significantly easier to evaluate. In one dimension the set of orthonormal polynomials are the normalised Legendre polynomials which are denoted by $\hat{P}_i(x)$. The L^2 optimal polynomial $p^\star(x) \in \mathcal{P}^3$ of $f(x)$ is therefore given by

$$p^\star(x) = \int_{-1}^{1} \hat{P}_i(y)f(y)\,\mathrm{d}y\,\hat{P}_i(x), \tag{2.28}$$

which has an L^2 error of 0.33. This represents a 17.5% decrease in error compared with a collocation projection using Gauss-Legendre points and a 51.5% decrease compared with a collocation projection using equispaced points.

Stability. The non-linear stability of FR type schemes depends on all projections being performed using the L^2 optimal polynomial [11, 33]. If another polynomial is used, such as one obtained from a collocation projection, then *aliasing* an occur. To illustrate this consider expanding $f(x)$ in terms of orthonormal basis functions

$$f(x) = \gamma_i \hat{P}_i(x) = p^\star(x) + \xi(x),$$

where $\xi(x)$ contains the modes of $f(x)$ that are not in the space of $p^\star(x)$. From the numerical experiments performed above it is known that, if a collocation projection is utilised, the resulting polynomial $p(x)$ may be different from $p^\star(x)$. If the polynomials are different then this implies that the modal expansion coefficients must also be different. This suggests that when using anything but the L^2 optimal expansion that there is the potential for modes which are not being resolved to impact—or alias—those which are being resolved.

Mitigation. In FR aliasing can be mitigated through one of three approaches.

1. From the one dimensional numerical experiments performed above it is clear that when performing a collocation projection the choice of sampling points can have a big impact on the L^2 error. The first approach is therefore to use combinations of solution and flux points which yield a better approximation to the L^2 optimal polynomial. Although determination of such points for non tensor product elements is a topic of active research [34–37] this approach has the advantage of not requiring any modifications to be made to the numerics. An in depth analysis of solution point placement inside of line segments and triangles is the topic of chapter 4.
2. It is known that the effects of aliasing are most pronounced in the highest frequency modes of the expansion [5, 38, 39]. A second means of stabilising the simulation is therefore to periodically filter the higher modes of the expansion [11, 40]. One means of accomplishing this is through an exponential filter. In

modal space this takes the form of a diagonal matrix in which

$$\Lambda_{e\sigma\sigma} = \begin{cases} 1 & \text{if } \deg \psi_{e\sigma} \leq \eta_c \\ \exp\Big(-\alpha((\deg \psi_{e\sigma} - \eta_c)/(\eta_m - \eta_c))^s \Big) & \text{otherwise,} \end{cases} \tag{2.29}$$

where $\alpha \approx \log \epsilon$, $\eta_c < \eta_m$ is a cutoff parameter, s is the strength of the filter, and $\eta_m = \max_\sigma \deg \psi_{e\sigma}$. Each entry indicates the amount of damping that should be applied to the $\psi_{e\sigma}$ mode of the solution. Damping is only applied to modes whose degree is greater than the cutoff with higher modes receiving progressively more damping. The rate at which this ramps up is controlled by s. Using the definition of the Vandermonde matrix the exponentially filtered solution in nodal space can be expressed as

$$\mathcal{F}_e u_{e\rho n\alpha} = \mathcal{V}^{-1}_{e\rho\nu} \Lambda_{e\nu\mu} \mathcal{V}_{e\mu\sigma} u_{e\sigma n\alpha}. \tag{2.30}$$

The computational costs associated with filtering are minimal. It is generally not necessary to filter the solution every time step and the kernel is completely element-local. However, unless care is taken when choosing the parameters it is very easy to introduce too much dissipation into the simulation; eliminating many of the advantages of high-order methods.

3. The final strategy is *anti-aliasing* where projections are performed by computing the coefficients in the modal expansion directly. This requires the evaluation of integrals of the form

$$\gamma_{e\sigma} = \int_{\tilde{\mathbf{\Omega}}_e} \psi_{e\sigma}(\tilde{\mathbf{x}}) f(\tilde{\mathbf{x}}) \, \mathrm{d}\tilde{\mathbf{x}},$$

where $f(\tilde{\mathbf{x}})$ is the transformed function being projected. These integrals are usually evaluated using Gaussian quadrature in which

$$\gamma_{e\sigma} \approx \omega^{(q)}_{e\rho} \psi_{e\sigma}(\tilde{\mathbf{x}}^{(q)}_{e\rho}) f(\tilde{\mathbf{x}}^{(q)}_{e\rho}), \tag{2.31}$$

where $\{\tilde{\mathbf{x}}^{(q)}_{e\rho}\}$ is a set of $N^{(q)}_e$ abscissa and $\{\omega^{(q)}_{e\rho}\}$ the set of associated weights. In order to be effective it is important that the chosen quadrature rule be of sufficient *strength* to accurately approximate the integrals. Otherwise, the quadrature itself

can be a source of aliasing. This generally results in there being significantly more quadrature points inside the volume of an element than solution points and, similarly, more quadrature points on each face than flux points. Anti-aliasing therefore comes at a non-trivial computational cost. The process of identifying efficient quadratures rules inside of a variety of domains is explored in chapter 3.

To illustrate the application of anti-aliasing consider the evaluation of the transformed flux $\tilde{\mathbf{f}}^{(u)}_{e\rho n\alpha}$ of (2.12). First it is necessary to interpolate both the solution and its gradients to the quadrature points

$$u^{(q)}_{e\sigma n\alpha} = u^{(u)}_{e\rho n\alpha} \ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}^{(q)}_{e\sigma}), \tag{2.32}$$

$$\mathbf{q}^{(q)}_{e\sigma n\alpha} = \mathbf{q}^{(u)}_{e\rho n\alpha} \ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}^{(q)}_{e\sigma}), \tag{2.33}$$

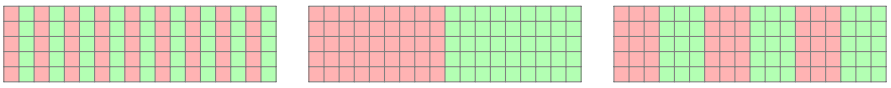
and using these the transformed flux at each quadrature point can be computed as

$$\tilde{\mathbf{f}}^{(q)}_{e\sigma n\alpha} = J^{(q)}_{e\sigma n} \mathbf{J}^{-1\,(q)}_{e\sigma n} \mathbf{f}_\alpha(u^{(q)}_{e\sigma n}, \mathbf{q}^{(q)}_{e\sigma n}),$$

where $J^{(q)}_{e\sigma n} = \det \mathbf{J}_{en}(\tilde{\mathbf{x}}^{(q)}_{e\sigma})$ and $\mathbf{J}^{-1\,(q)}_{e\sigma n} = \mathbf{J}^{-1}_{en}(\tilde{\mathbf{x}}^{(q)}_{e\sigma})$. With this the transformed flux at the solution points can be obtained by evaluating

$$\tilde{\mathbf{f}}^{(u)}_{e\rho n\alpha} = \psi_{e\nu}(\tilde{\mathbf{x}}^{(u)}_{e\rho}) \omega^{(q)}_{e\sigma} \psi_{e\nu}(\tilde{\mathbf{x}}^{(q)}_{e\sigma}) \tilde{\mathbf{f}}^{(q)}_{e\sigma n\alpha}. \tag{2.34}$$

A similar procedure can be used to anti-alias the transformed common normal flux $\mathfrak{F}_\alpha \tilde{f}^{(f_\perp)}_{e\sigma n\alpha}$ on the surface of each element. Rather than integrating over the interior of the reference element it is instead necessary to evaluate integrals over each of its faces. To accomplish this a quadrature rule is associated with each of the different face types in the simulation; line segments in two dimensions and triangles and quadrilaterals in three dimensions. Since the abscissa will be used for computing common quantities at interfaces it is imperative—just as with the flux points—that they are symmetric. These points are then projected onto the faces of each reference element to yield the set of flux quadrature points $\{\tilde{\mathbf{x}}^{(fq)}_{e\rho}\}$. Expressions similar to (2.32) and (2.33) can then be used to interpolate the solution and its gradient to these points. The transformed common normal flux can then be evaluated and, using a face-centric variant of (2.34), projected



(a) AoS. (b) SoA. (c) AoSoA($k = 3$)

Figure 2.3. Packing methodologies for $N_v = 2$ and $|\mathbf{\Omega}_e| = 9$.

onto the flux points. Here, however, the projection is performed on a per-face basis.

2.5 Matrix Representation

It is possible to cast the majority of operations in an FR step as matrix-matrix multiplications of the form

$$\mathbf{C} \leftarrow c_1 \mathbf{A}\mathbf{B} + c_2 \mathbf{C}, \tag{2.35}$$

where $c_{1,2}$ are constants, $\mathbf{A}$ is a constant operator matrix, and $\mathbf{B}$ and $\mathbf{C}$ are state matrices. To accomplish this the following constant operator matrix is introduced

$$\left(\mathbf{M}_e^0\right)_{\sigma\rho} = \ell_{e\rho}^{(u)}(\tilde{\mathbf{x}}_{e\sigma}^{(f)}), \qquad \dim \mathbf{M}_e^0 = N_e^{(f)} \times N_e^{(u)},$$

and the following state matrices

$$\left(\mathbf{U}_e^{(u)}\right)_{\rho(n\alpha)} = u_{e\rho n\alpha}^{(u)}, \qquad \dim \mathbf{U}_e^{(u)} = N_e^{(u)} \times N_V |\mathbf{\Omega}_e|,$$
$$\left(\mathbf{U}_e^{(f)}\right)_{\sigma(n\alpha)} = u_{e\sigma n\alpha}^{(f)}, \qquad \dim \mathbf{U}_e^{(f)} = N_e^{(f)} \times N_V |\mathbf{\Omega}_e|.$$

In specifying the solution matrices there is a degree of freedom regarding how the field variables of the various elements are packed along a row. The packing of field variables can be characterised by considering the distance, Δj, in columns between two subsequent field variables for a given element. The case of $\Delta j = 1$ corresponds to the array of structures (AoS) packing whereas the choice of $\Delta j = |\mathbf{\Omega}_e|$ leads to the structure of arrays (SoA) packing. A hybrid approach wherein $\Delta j = k$ with k being constant results in the AoSoA(k) approach. Illustrations of these three approaches can be seen in

Figure 2.3. An implementation is free to chose between any of these counting patterns so long as it is consistent. Using these matrices (2.6) can be reformulated as

$$\mathbf{U}_e^{(f)} = \mathbf{M}_e^0 \mathbf{U}_e^{(u)}. \tag{2.36}$$

In order to apply a similar procedure to (2.9) it is first necessary to let

$$\begin{aligned}
\left(\mathbf{M}_e^4\right)_{\rho\sigma} &= \left[\tilde{\boldsymbol{\nabla}} \ell_{e\rho}^{(u)}(\tilde{\mathbf{x}})\right]_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}_{e\sigma}^{(u)}}, & \dim \mathbf{M}_e^4, &= N_D N_e^{(u)} \times N_e^{(u)}, \\
\left(\mathbf{M}_e^6\right)_{\rho\sigma} &= \left[\hat{\tilde{\mathbf{n}}}_{e\rho}^{(f)} \cdot \tilde{\boldsymbol{\nabla}} \cdot \mathbf{g}_{e\rho}^{(f)}(\tilde{\mathbf{x}})\right]_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}_{e\sigma}^{(f)}}, & \dim \mathbf{M}_e^6, &= N_D N_e^{(u)} \times N_e^{(f)}, \\
\left(\mathbf{C}_e^{(f)}\right)_{\rho(n\alpha)} &= \mathfrak{C}_\alpha u_{e\rho n\alpha}^{(f)}, & \dim \mathbf{C}_e^{(f)} &= N_e^{(f)} \times N_V |\boldsymbol{\Omega}_e|, \\
\left(\tilde{\mathbf{Q}}_e^{(u)}\right)_{\sigma(n\alpha)} &= \tilde{\mathbf{q}}_{e\sigma n\alpha}^{(u)}, & \dim \tilde{\mathbf{Q}}_e^{(u)} &= N_D N_e^{(u)} \times N_V |\boldsymbol{\Omega}_e|,
\end{aligned}$$

Here it is important to qualify assignments of the form $\mathbf{A}_{ij} = \mathbf{x}$ where $\mathbf{x}$ is a N_D component vector. As above there is a degree of freedom associated with the packing. With the benefit of foresight the stride between subsequent elements of $\mathbf{x}$ in a matrix column is taken to be either $\Delta i = N_e^{(u)}$ or $\Delta i = N_e^{(f)}$ depending on the context. Hence (2.9) reduces to

$$\begin{aligned}
\tilde{\mathbf{Q}}_e^{(u)} &= \mathbf{M}_e^6 \left\{ \mathbf{C}_e^{(f)} - \mathbf{U}_e^{(f)} \right\} + \mathbf{M}_e^4 \mathbf{U}_e^{(u)} \\
&= \mathbf{M}_e^6 \left\{ \mathbf{C}_e^{(f)} - \mathbf{M}_e^0 \mathbf{U}_e^{(u)} \right\} + \mathbf{M}_e^4 \mathbf{U}_e^{(u)} \\
&= \mathbf{M}_e^6 \mathbf{C}_e^{(f)} + \left\{ \mathbf{M}_e^4 - \mathbf{M}_e^6 \mathbf{M}_e^0 \right\} \mathbf{U}_e^{(u)}.
\end{aligned} \tag{2.37}$$

Applying the procedure to (2.11)

$$\begin{aligned}
\mathbf{M}_e^5 &= \operatorname{diag}(\mathbf{M}_e^0, \ldots, \mathbf{M}_e^0) & \dim \mathbf{M}_e^5 &= N_D N_e^{(f)} \times N_D N_e^{(u)}, \\
\left(\mathbf{Q}_e^{(u)}\right)_{\sigma(n\alpha)} &= \mathbf{q}_{e\sigma n\alpha}^{(u)}, & \dim \mathbf{Q}_e^{(u)} &= N_D N_e^{(u)} \times N_V |\boldsymbol{\Omega}_e|, \\
\left(\mathbf{Q}_e^{(f)}\right)_{\sigma(n\alpha)} &= \mathbf{q}_{e\sigma n\alpha}^{(f)}, & \dim \mathbf{Q}_e^{(f)} &= N_D N_e^{(f)} \times N_V |\boldsymbol{\Omega}_e|,
\end{aligned}$$

hence

$$\mathbf{Q}_e^{(f)} = \mathbf{M}_e^5 \mathbf{Q}_e^{(u)}, \tag{2.38}$$

where $\mathbf{M}_e^5$ can be seen to be block diagonal. This is a direct consequence of the above

choices for Δi. Finally, to rewrite (2.18)

$$\begin{aligned}
\left(\mathbf{M}_e^1\right)_{\rho\sigma} &= \left[\tilde{\nabla}\ell_{e\rho}^{(u)}(\tilde{\mathbf{x}})\right]^T_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}_{e\sigma}^{(u)}}, & \dim \mathbf{M}_e^1 &= N_e^{(u)} \times N_D N_e^{(u)},\\
\left(\mathbf{M}_e^2\right)_{\rho\sigma} &= \left[\ell_{e\rho}^{(u)}(\tilde{\mathbf{x}}_{e\sigma}^{(f)})\hat{\mathbf{n}}_{e\sigma}^{(f)}\right]^T, & \dim \mathbf{M}_e^2 &= N_e^{(f)} \times N_D N_e^{(u)},\\
\left(\mathbf{M}_e^3\right)_{\rho\sigma} &= \left[\tilde{\nabla}\cdot\mathbf{g}_{e\sigma}^{(f)}(\tilde{\mathbf{x}})\right]_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}_{e\rho}^{(u)}}, & \dim \mathbf{M}_e^3 &= N_e^{(u)} \times N_e^{(f)},\\
\left(\tilde{\mathbf{D}}_e^{(f)}\right)_{\sigma(n\alpha)} &= \mathfrak{F}_\alpha \tilde{f}_{e\sigma n\alpha}^{(f_\perp)}, & \dim \tilde{\mathbf{D}}_e^{(f)} &= N_e^{(f)} \times N_V|\mathbf{\Omega}_e|,\\
\left(\tilde{\mathbf{F}}_e^{(u)}\right)_{\rho(n\alpha)} &= \tilde{\mathbf{f}}_{e\rho n\alpha}^{(u)}, & \dim \tilde{\mathbf{F}}_e^{(u)} &= N_D N_e^{(u)} \times N_V|\mathbf{\Omega}_e|,\\
\left(\tilde{\mathbf{R}}_e^{(u)}\right)_{\rho(n\alpha)} &= (\tilde{\nabla}\cdot\tilde{\mathbf{f}})_{e\rho n\alpha}^{(u)}, & \dim \tilde{\mathbf{R}}_e^{(u)} &= N_e^{(u)} \times N_V|\mathbf{\Omega}_e|,
\end{aligned}$$

and after substitution of (2.13) for $\tilde{f}_{e\sigma n\alpha}^{(f_\perp)}$ obtain

$$\begin{aligned}
\tilde{\mathbf{R}}_e^{(u)} &= \mathbf{M}_e^3\left\{\tilde{\mathbf{D}}_e^{(f)} - \mathbf{M}_e^2\tilde{\mathbf{F}}_e^{(u)}\right\} + \mathbf{M}_e^1\tilde{\mathbf{F}}_e^{(u)}\\
&= \mathbf{M}_e^3\tilde{\mathbf{D}}_e^{(f)} + \left\{\mathbf{M}_e^1 - \mathbf{M}_e^3\mathbf{M}_e^2\right\}\tilde{\mathbf{F}}_e^{(u)}.
\end{aligned} \tag{2.39}$$

Anti-aliasing. The two core operations associated with anti-aliasing, interpolations to quadrature points and projections from them, can be readily cast as matrix multiplications. When anti-aliasing the transformed flux (2.32) can be rewritten by introducing

$$\begin{aligned}
\left(\mathbf{M}_e^7\right)_{\sigma\rho} &= \ell_{e\rho}^{(u)}(\tilde{\mathbf{x}}_{e\sigma}^{(q)}), & \dim \mathbf{M}_e^7 &= N_e^{(q)} \times N_e^{(u)},\\
\left(\mathbf{U}_e^{(q)}\right)_{\rho(n\alpha)} &= u_{e\rho n\alpha}^{(q)}, & \dim \mathbf{U}_e^{(q)} &= N_e^{(q)} \times N_V|\mathbf{\Omega}_e|,
\end{aligned}$$

hence

$$\mathbf{U}_e^{(q)} = \mathbf{M}_e^7\mathbf{U}_e^{(u)}. \tag{2.40}$$

Similarly, (2.33) can be rewritten by letting

$$\begin{aligned}
\mathbf{M}_e^{10} &= \mathrm{diag}(\mathbf{M}_e^7, \ldots, \mathbf{M}_e^7), & \dim \mathbf{M}_e^{10} &= N_D N_e^{(q)} \times N_D N_e^{(u)},\\
\left(\mathbf{Q}_e^{(q)}\right)_{\sigma(n\alpha)} &= \mathbf{q}_{e\sigma n\alpha}^{(q)}, & \dim \mathbf{Q}_e^{(q)} &= N_D N_e^{(q)} \times N_V|\mathbf{\Omega}_e|,
\end{aligned}$$

where $\mathbf{M}_e^{10}$ is observed to have a block diagonal structure similar to that of $\mathbf{M}_e^5$. Using these it follows that

$$\mathbf{Q}_e^{(q)} = \mathbf{M}_e^{10}\mathbf{Q}_e^{(u)}. \tag{2.41}$$

The projection of the transformed flux in (2.34) can be brought into this framework by defining

$$\left(\mathbf{M}_e^9\right)_{\rho\sigma} = \psi_{e\nu}(\tilde{\mathbf{x}}_{e\rho}^{(u)})\omega_{e\sigma}^{(q)}\psi_{e\nu}(\tilde{\mathbf{x}}_{e\sigma}^{(q)}), \qquad \dim \mathbf{M}_e^9 = N_e^{(u)} \times N_e^{(q)},$$
$$\left(\tilde{\mathbf{F}}_e^{(q)}\right)_{\rho(n\alpha)} = \tilde{\mathbf{f}}_{e\rho n\alpha}^{(q)}, \qquad \dim \tilde{\mathbf{F}}_e^{(q)} = N_D N_e^{(q)} \times N_V|\mathbf{\Omega}_e|,$$

such that (2.39) now reads

$$\tilde{\mathbf{R}}_e^{(u)} = \mathbf{M}_e^3\tilde{\mathbf{D}}_e^{(f)} + \left\{\mathbf{M}_e^1 - \mathbf{M}_e^3\mathbf{M}_e^2\right\}\mathbf{M}_e^9\tilde{\mathbf{F}}_e^{(q)}. \tag{2.42}$$

As a means of improving efficiency it is observed that when anti-aliasing it is necessary to interpolate the solution to both the flux points using $\mathbf{M}_e^0$ and to the quadrature points using $\mathbf{M}_e^7$. By arranging the storage for $\mathbf{U}_e^{(u)}$ and $\mathbf{U}_e^{(q)}$ carefully it is possible to define $\mathbf{M}_e^8 = (\mathbf{M}_e^0 \mid \mathbf{M}_e^7)$ which performs both interpolations in a single step.

The introduction of surface anti-aliasing into the matrix formulation is more subtle. Since no additional operations are required it is possible to perform surface anti-aliasing through a redefinition of the existing operator/state matrices. Specifically, with the exceptions of $\mathbf{M}_e^3$ and $\mathbf{M}_e^6$, all references to the flux points $\tilde{\mathbf{x}}_{e\sigma}^{(f)}$ are replaced by the flux quadrature points $\tilde{\mathbf{x}}_{e\sigma}^{(fq)}$. Hence, all operations that previously were performed at the flux points, such as computing the common interface solution and common normal fluxes, are now performed at the flux quadrature points. Next, $\mathbf{M}_e^3$ and $\mathbf{M}_e^6$ are post-multiplied by a matrix that performs an L^2 projection from the flux quadrature points on each face to the true flux points on each face. As the projection is done on a per-face basis this matrix has a block diagonal structure. The definition of each block is similar to that of $\mathbf{M}_e^9$ except that e now refers to a face type on the reference element.

Filtering. Applying the standard procedure to the filtering equation (2.30)

$$\left(\mathbf{V}_e^{(u)}\right)_{\rho(n\alpha)} = \mathcal{F}_e u_{e\rho n\alpha}^{(u)}, \qquad \dim \mathbf{V}_e^{(u)} = N_e^{(u)} \times N_V|\mathbf{\Omega}_e|,$$
$$\left(\mathbf{M}_e^{11}\right)_{\rho\sigma} = \mathcal{V}_{e\rho\nu}^{-1}\Lambda_{e\nu\mu}\mathcal{V}_{e\mu\sigma}, \qquad \dim \mathbf{M}_e^{11} = N_e^{(u)} \times N_e^{(u)},$$

hence

$$\mathbf{V}_e^{(u)} = \mathbf{M}_e^{11}\mathbf{U}_e^{(u)}. \tag{2.43}$$

2.6 Governing Systems

Euler equations. Using the framework introduced in §2.1 the three dimensional Euler equations can be expressed in conservative form as

$$u = \begin{Bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ E \end{Bmatrix}, \qquad \mathbf{f} = \mathbf{f}^{(\text{inv})} = \begin{Bmatrix} \rho v_x & \rho v_y & \rho v_z \\ \rho v_x^2 + p & \rho v_y v_x & \rho v_z v_x \\ \rho v_x v_y & \rho v_y^2 + p & \rho v_z v_y \\ \rho v_x v_z & \rho v_y v_z & \rho v_z^2 + p \\ v_x(E+p) & v_y(E+p) & v_z(E+p) \end{Bmatrix}, \tag{2.44}$$

with u and $\mathbf{f}$ together satisfying (2.1). In the above ρ is the mass density of the fluid, $\mathbf{v} = (v_x, v_y, v_z)^T$ is the fluid velocity vector, E is the total energy per unit volume, and p is the pressure. For a perfect gas the pressure and total energy can be related by the ideal gas law

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho \|\mathbf{v}\|^2, \tag{2.45}$$

where $\gamma = c_p/c_v$. Observe here the presence of terms of the form $\rho v_i v_j$ in $\mathbf{f}$. Evaluating these terms as a function of the conservative variables requires taking the quotient of $\rho v_j/\rho$. However, in general, the quotient of two polynomials is not itself a polynomial. Hence, the Euler flux function is not just non-linear but also non-polynomial.

With the fluxes specified all that remains is to prescribe a method for computing the common normal flux $\mathfrak{F}_\alpha$ at interfaces as defined in (2.15). This can be accomplished using an approximate Riemann solver for the Euler equations. There exist a variety of such solvers as detailed in [41] and Appendix A.

Compressible Navier–Stokes equations. The compressible Navier–Stokes equations can be viewed as an extension of the Euler equations via the inclusion of viscous terms. Within the framework outlined above the flux now takes the form of

$\mathbf{f} = \mathbf{f}^{(\text{inv})} - \mathbf{f}^{(\text{vis})}$ where

$$\mathbf{f}^{(\text{vis})} = \begin{Bmatrix} 0 & 0 & 0 \\ \mathcal{T}_{xx} & \mathcal{T}_{yx} & \mathcal{T}_{zx} \\ \mathcal{T}_{xy} & \mathcal{T}_{yy} & \mathcal{T}_{zy} \\ \mathcal{T}_{xz} & \mathcal{T}_{yz} & \mathcal{T}_{zz} \\ v_i\mathcal{T}_{ix} + \Delta\partial_x T & v_i\mathcal{T}_{iy} + \Delta\partial_y T & v_i\mathcal{T}_{iz} + \Delta\partial_z T \end{Bmatrix}. \tag{2.46}$$

In the above $\Delta = \mu c_p / P_r$ where μ is the dynamic viscosity and P_r is the Prandtl number. The components of the stress-energy tensor are given by

$$\mathcal{T}_{ij} = \mu(\partial_i v_j + \partial_j v_i) - \frac{2}{3}\mu\delta_{ij}\boldsymbol{\nabla}\cdot\mathbf{v}. \tag{2.47}$$

Using the ideal gas law the temperature can be expressed as

$$T = \frac{1}{c_v}\frac{1}{\gamma - 1}\frac{p}{\rho}, \tag{2.48}$$

with partial derivatives thereof being given according to the quotient rule.

Since the Navier–Stokes equations are an advection-diffusion type system it is necessary to both compute a common solution at element boundaries and augment the inviscid Riemann solver to handle the viscous part of the flux. A popular approach is the LDG method as presented in [11, 26]. In this approach the common solution is given $\forall\alpha$ according to

$$\mathfrak{C}(u_L, u_R) = (\tfrac{1}{2} - \beta)u_L + (\tfrac{1}{2} + \beta)u_R, \tag{2.49}$$

where β controls the degree of upwinding/downwinding. The common normal interface flux is then prescribed, once again $\forall\alpha$, according to

$$\mathfrak{F}(u_L, u_R, \mathbf{q}_L, \mathbf{q}_R, \hat{\mathbf{n}}_L) = \mathfrak{F}^{(\text{inv})} - \mathfrak{F}^{(\text{vis})}, \tag{2.50}$$

where $\mathfrak{F}^{(\text{inv})}$ is a suitable inviscid Riemann solver (see Appendix A) and

$$\mathfrak{F}^{(\text{vis})} = \hat{\mathbf{n}}_L \cdot \left\{(\tfrac{1}{2} + \beta)\mathbf{f}_L^{(\text{vis})} + (\tfrac{1}{2} - \beta)\mathbf{f}_R^{(\text{vis})}\right\} + \tau(u_L - u_R), \tag{2.51}$$

with τ being a penalty parameter, $\mathbf{f}_L^{(\text{vis})} = \mathbf{f}^{(\text{vis})}(u_L, \mathbf{q}_L)$, and $\mathbf{f}_R^{(\text{vis})} = \mathbf{f}^{(\text{vis})}(u_R, \mathbf{q}_R)$. It is observed here that if the common solution is upwinded then the common normal flux will be downwinded. Generally, $\beta = \pm 1/2$ as this results in the numerical scheme having a compact stencil and $0 \leq \tau \leq 1$.

Presentation in two dimensions. The above prescriptions of the Euler and Navier–Stokes equations are valid for the case of $N_D = 3$. The two dimensional formulation can be recovered by deleting the fourth rows in the definitions of u, $\mathbf{f}^{(\text{inv})}$ and $\mathbf{f}^{(\text{vis})}$ along with the third columns of $\mathbf{f}^{(\text{inv})}$ and $\mathbf{f}^{(\text{vis})}$. Vectors are now two dimensional with the velocity being given by $\mathbf{v} = (v_x, v_y)^T$.

Chapter 3

Quadrature Rules

When using FR in conjunction with anti-aliasing it is necessary to evaluate integrals inside of the standard elements. A popular numerical integration technique is that of Gaussian quadrature in which

$$\int_{\mathbf{\Omega}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \sum_{i}^{N_p} \omega_i f(\mathbf{x}_i), \tag{3.1}$$

where $f(\mathbf{x})$ is the function to be integrated, $\{\mathbf{x}_i\}$ are a set of N_p points, and $\{\omega_i\}$ the set of associated weights. The points and weights are said to define a *quadrature rule*. A rule is said to be of strength ϕ if it is capable of exactly integrating any polynomial of maximal degree ϕ over $\mathbf{\Omega}$. A degree ϕ polynomial $p(\mathbf{x})$ with $\mathbf{x} \in \mathbf{\Omega}$ can be expressed as a linear combination of basis polynomials

$$p(\mathbf{x}) = \sum_{i}^{|\mathcal{P}^\phi|} \alpha_i \mathcal{P}_i^\phi(\mathbf{x}), \qquad \alpha_i = \int_{\mathbf{\Omega}} p(\mathbf{x}) \mathcal{P}_i^\phi(\mathbf{x})\,\mathrm{d}\mathbf{x}, \tag{3.2}$$

where $\mathcal{P}^\phi$ is the set of basis polynomials of degree $\leq \phi$ whose cardinality is given by $|\mathcal{P}^\phi|$. From the linearity of integration it therefore follows that a strength ϕ quadrature rule is one which can exactly integrate the basis. Taking $f \in \mathcal{P}^\phi$ the task of obtaining an N_p point quadrature rule of strength ϕ is hence reduced to finding a solution to a system of $|\mathcal{P}^\phi|$ non-linear equations. This system can be seen to possess $(N_D + 1)N_p$ degrees of freedom where $N_D \geq 2$ corresponds to the number of spatial dimensions.

In the case of $N_p \lesssim 10$ the above system can often be solved analytically using a computer algebra package. However, beyond this it is usually necessary to solve the above system—or a simplification thereof—numerically. Much of the research into multidimensional quadrature over the past five decades has been directed towards the

development of such numerical methods. The traditional objective when constructing quadrature rules is to obtain a rule of strength ϕ inside of a domain $\mathbf{\Omega}$ using the fewest number of points. To this end efficient quadrature rules have been derived for a variety of domains: triangles [36, 37, 42–49], quadrilaterals [49–51], tetrahedra [45, 47, 52, 53], prisms [54], pyramids [55], and hexahedra [49, 56–58]. For finite element applications it is desirable that (i) points are arranged symmetrically inside of the domain, (ii) all of the points are strictly inside of the domain, and (iii) all of the weights are positive. The consideration given to these criteria in the literature cited above depends strongly on the intended field of application—not all rules are derived with finite element methods in mind.

Much of the existing literature is predicated on the assumption that the integrand sits in the space of $\mathcal{P}^{\phi}$. Under this assumption there is little, other than the criteria listed above, to distinguish two N_p rules of strength ϕ; both can be expected to compute the integral exactly with the same number of functional evaluations. It is therefore common practice to terminate the rule discovery process as soon as a rule is found. However, there are cases when either the integrand is inherently non-polynomial in nature, e.g. the quotient of two polynomials, or of a high degree, e.g. a polynomial raised to a high power. In these circumstances the above assumption no longer holds and it is necessary to consider the truncation term associated with each rule. Hence, within this context it is no longer clear that the traditional objective of minimising the number of points required to obtain a rule of given strength is suitable: it is possible that the addition of an extra point will permit the integration of several of the basis functions of degree $\phi + 1$.

3.1 Basis Polynomials

The defining property of a quadrature rule for a domain $\mathbf{\Omega}$ is its ability to exactly integrate the set of basis polynomials, $\mathcal{P}^{\phi}$. This set has an infinite number of representations the simplest of which being the monomials. In two dimensions the monomials

can be expressed as

$$\mathcal{P}^{\phi} = \left\{ x^i y^j \mid 0 \leq i \leq \phi,\ 0 \leq j \leq \phi - i \right\}, \tag{3.3}$$

where ϕ is the maximal degree. Unfortunately, at higher degrees the monomials become extremely sensitive to small perturbations in the inputs. This gives rise to polynomial systems which are poorly conditioned and hence difficult to solve numerically [47, 52]. A solution to this is to switch to an orthonormal basis set defined in two dimensions as

$$\mathcal{P}^{\phi} = \left\{ \psi_{(ij)}(\mathbf{x}) \mid 0 \leq i \leq \phi,\ 0 \leq j \leq \phi - i \right\}, \tag{3.4}$$

where $\mathbf{x} = (x, y)^T$ and $\psi_{(ij)}(\mathbf{x})$ satisfies the familiar orthonormality property $\forall \mu, \nu$

$$\int_{\mathbf{\Omega}} \psi_{(ij)}(\mathbf{x}) \psi_{(\mu\nu)}(\mathbf{x}) \, \mathrm{d}\mathbf{x} = \delta_{i\mu}\delta_{j\nu}. \tag{3.5}$$

In addition to being exceptionally well conditioned, orthonormal polynomial bases have other useful properties. Taking the constant mode of the basis to be $\psi_{(00)}(\mathbf{x}) = 1/c$ it follows that

$$\int_{\mathbf{\Omega}} \psi_{(ij)}(\mathbf{x}) \, \mathrm{d}\mathbf{x} = c \int_{\mathbf{\Omega}} \psi_{(00)}(\mathbf{x}) \psi_{(ij)}(\mathbf{x}) \, \mathrm{d}\mathbf{x} = c\delta_{i0}\delta_{j0}, \tag{3.6}$$

from which it can be deduced that all non-constant modes of the basis integrate up to zero. Following Witherden and Vincent [37] this property is used to define the truncation error associated with an N_p point rule

$$\xi^2(\phi) = \sum_{i,j} \left\{ \sum_{k}^{N_p} \omega_k \psi_{(ij)}(\mathbf{x}_k) - c\delta_{i0}\delta_{j0} \right\}^2, \tag{3.7}$$

This definition is convenient as it is free from both integrals and normalisation factors. The task of constructing an N_p point quadrature rule of strength ϕ is synonymous with finding a set of points and weights that minimise $\xi(\phi)$.

Although the above discussion has been presented primarily in two dimensions all of the ideas and relations carry over into three dimensions.

3.2 Symmetry Orbits

A symmetric arrangement of N_p points inside of a reference domain can be decomposed into a linear combination of *symmetry orbits*. This concept is best elucidated with an example. Consider a line segment defined by $[-1, 1]$. The segment possesses two symmetries: an identity transformation and a reflection about the origin. For an arrangement of distinct points to be symmetric it follows that if there is a point at α where $0 < \alpha \leq 1$ there must also be a point at $-\alpha$. This can be codified by writing $\mathcal{S}_2(\alpha) = \pm\alpha$ with $|\mathcal{S}_2| = 2$. The function $\mathcal{S}_2$ is an example of a *symmetry orbit* that takes a single *orbital parameter*, α, and generates two distinct points. In the limit of $\alpha \to 0$ the two points become degenerate. This degeneracy can be handled by introducing a second orbit, $\mathcal{S}_1 = 0$, with $|\mathcal{S}_1| = 1$. Having identified the symmetries it is now possible to decompose a symmetric arrangement of points as

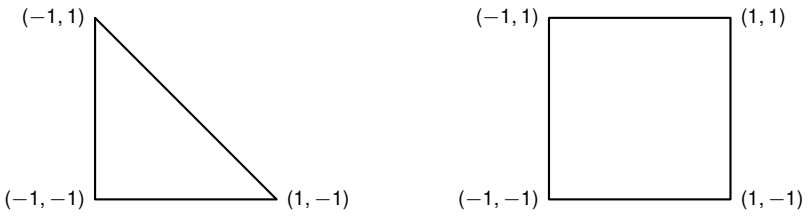
$$N_p = n_1|\mathcal{S}_1| + n_2|\mathcal{S}_2| = n_1 + 2n_2,$$

where $n_1 \in \{0, 1\}$ and $n_2 \geq 0$ with the constraint on n_1 being necessary to ensure uniqueness. This is a constrained linear Diophantine equation; albeit one that is trivially solvable and admits only a single solution. As a concrete example consider $N_p = 11$. Solving the above equation it is found that $n_1 = 1$ and $n_2 = 5$. The n_1 orbit does not take any arguments and so does not contribute any degrees of freedom. Each n_2 orbit takes a single parameter, α, and so contributes one degree of freedom for a grand total of five. This is less than half that associated with the asymmetrical case. Hence, by parameterising the problem in terms of symmetry orbits it is possible to simultaneously reduce the number of degrees of freedom while guaranteeing a symmetric distribution of points.

Symmetries also serve to reduce the number of basis polynomials that must be considered when computing $\xi(\phi)$. Consider the following two monomials

$$p_1(x, y) = x^i y^j, \qquad \text{and} \qquad p_2(x, y) = x^j y^i,$$

defined inside of a square domain with vertices $(-1, -1)$ and $(1, 1)$. From these definitions it is evident that $p_1(x, y) = p_2(y, x)$. As this is a symmetry which is expressed by



(a) Triangle. **(b)** Quadrilateral.

Figure 3.1. Reference domains in two dimensions.

the domain it is clear that any symmetric quadrature rule capable of integrating p_1 is also capable of integrating p_2. Further, when the index i is odd $p_1(x, y) = -p_1(-x, y)$. Similarly, when j is odd $p_1(x, y) = -p_1(x, -y)$. In both cases it follows that the integral of p_1 is zero over the domain. More importantly, it also follows that *any* set of symmetric points are also capable of obtaining this result. This is due to terms on the right hand side of (3.1) pairing up and cancelling out. A consequence of this is that not all of the equations in the system specified by (3.1) are independent. Having identified such polynomials for a given domain it is legitimate to exclude them from the definition of $\xi(\phi)$. Although this exclusion does change the value of $\xi(\phi)$ in the case of a non-zero truncation error the effect is not significant. The set of basis polynomials which *are* included is termed the *objective basis*, and shall denote this by $\tilde{\mathcal{P}}^{\phi}$.

3.3 Reference Domains

In the paragraphs which follow $\hat{P}_i^{(\alpha,\beta)}(x)$ is taken to refer to a *normalised* Jacobi polynomial as specified in §18.3 of [59]. In two dimensions the coordinates axes are defined as $\mathbf{x} = (x, y)$ and as $\mathbf{x} = (x, y, z)$ in three dimensions.

Triangle. The reference triangle can be seen in Figure 3.1 and has an area given by $\int_{-1}^{1}\int_{-1}^{-y} \mathrm{d}x\,\mathrm{d}y = 2$. A triangle has six symmetries: two rotations, three reflections, and the identity transformation. A simple means of realising these symmetries is to

transform into barycentric coordinates

$$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3)^T \quad 0 \leq \lambda_i \leq 1, \lambda_1 + \lambda_2 + \lambda_3 = 1, \tag{3.8}$$

which are related to Cartesian coordinates via

$$\mathbf{x} = \begin{pmatrix} -1 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix} \boldsymbol{\lambda}, \tag{3.9}$$

where the columns of the matrix can be seen to be the vertices of the reference triangle. The utility of barycentric coordinates is that the symmetric counterparts to a point $\boldsymbol{\lambda}$ are given by its unique permutations. The number of unique permutations depends on the number of distinct components of $\boldsymbol{\lambda}$ and leads us to the following three symmetry orbits

$$\begin{aligned} \mathcal{S}_1 &= (\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}), & |\mathcal{S}_1| &= 1, \\ \mathcal{S}_2(\alpha) &= \text{Perm}(\alpha, \alpha, 1 - 2\alpha), & |\mathcal{S}_2| &= 3, \\ \mathcal{S}_3(\alpha, \beta) &= \text{Perm}(\alpha, \beta, 1 - \alpha - \beta), & |\mathcal{S}_3| &= 6, \end{aligned}$$

where α and β are suitably constrained as to ensure that the resulting coordinates are inside of the domain.

It can be easily verified that the orthonormal polynomial basis inside of the reference triangle is given by

$$\psi_{(ij)}(\mathbf{x}) = \sqrt{2}\hat{P}_i(a)\hat{P}_j^{(2i+1,0)}(b)(1-b)^i, \tag{3.10}$$

where $a = 2(1 + x)/(1 - y) - 1$, and $b = y$ with the objective basis being given by

$$\tilde{\mathcal{P}}^{\phi} = \left\{ \psi_{(ij)}(\mathbf{x}) \mid 0 \leq i \leq \phi,\ i \leq j \leq \phi - i \right\}. \tag{3.11}$$

In the asymptotic limit the cardinality of the objective basis is half that of the complete basis. However, the modes of this objective basis are known not to be completely independent. Several authors have investigated the derivation of an optimal quadrature basis on the triangle. Details can be found in the papers of Lyness [42] and Dunavant [43].

Quadrilateral. The reference quadrilateral can be seen in Figure 3.1. The area is simply $\int_{-1}^{1}\int_{-1}^{1} \mathrm{d}x\,\mathrm{d}y = 4$. A square has eight symmetries: three rotations, four reflections and the identity transformation. Applying these symmetries to a point (α,β) with $0 \leq (\alpha,\beta) \leq 1$ will yield a set $\chi(\alpha,\beta)$ containing its counterparts. The cardinality of χ depends on if any of the symmetries give rise to identical points. This can be seen to occur when either $\beta = \alpha$ or $\beta = 0$. Enumerating the possible combinations of the above conditions gives rise to the following four symmetry orbits

$$\begin{aligned} \mathcal{S}_1 &= (0,0), & |\mathcal{S}_1| &= 1, \\ \mathcal{S}_2(\alpha) &= \chi(\alpha,0), & |\mathcal{S}_2| &= 4, \\ \mathcal{S}_3(\alpha) &= \chi(\alpha,\alpha), & |\mathcal{S}_3| &= 4, \\ \mathcal{S}_4(\alpha,\beta) &= \chi(\alpha,\beta), & |\mathcal{S}_4| &= 8. \end{aligned}$$

Trivially, the orthonormal basis inside of the quadrilateral is given by

$$\psi_{(ij)}(\mathbf{x}) = \hat{P}_i(a)\hat{P}_j(b), \tag{3.12}$$

where $a = x$, and $b = y$. The objective basis is found to be

$$\tilde{\mathcal{P}}^{\phi} = \left\{\psi_{(ij)}(\mathbf{x}) \mid 0 \leq i \leq \phi,\ i \leq j \leq \phi - i,\ (i,j) \text{ even}\right\}, \tag{3.13}$$

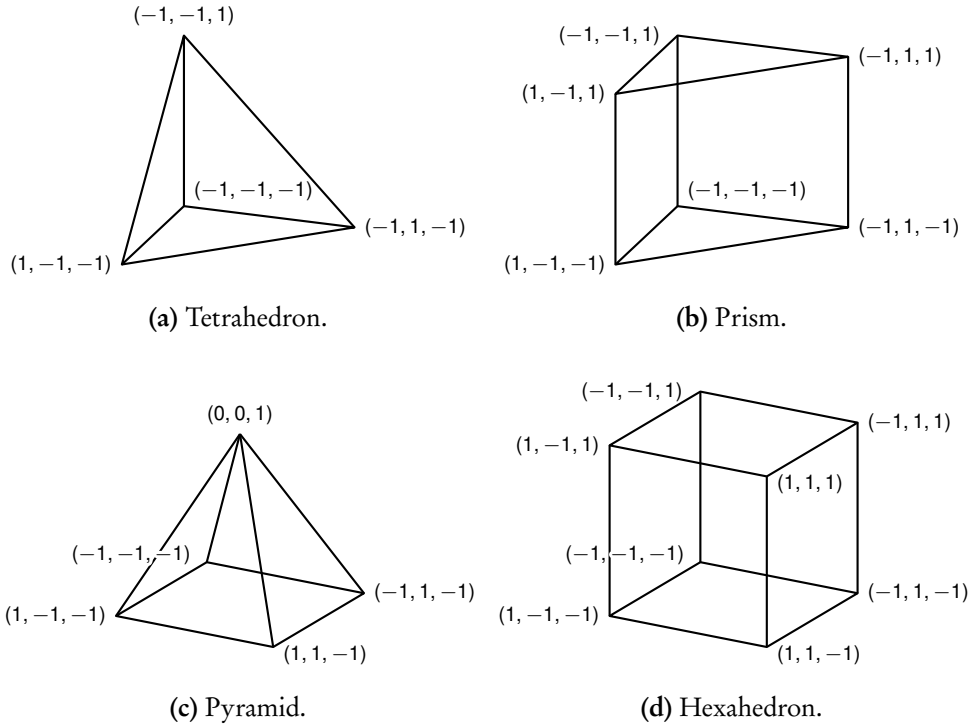
with a cardinality one eighth that of the complete basis.

Tetrahedron. The reference tetrahedron is a right-tetrahedron as depicted in Figure 3.2. Integrating up the volume it is found that $\int_{-1}^{1}\int_{-1}^{-z}\int_{-1}^{-1-y-z} \mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = 4/3$. A tetrahedron has a total of 24 symmetries. Once again it is convenient to work in terms of barycentric coordinates which are specified for a tetrahedron as

$$\boldsymbol{\lambda} = (\lambda_1,\lambda_2,\lambda_3,\lambda_4)^T \quad 0 \leq \lambda_i \leq 1, \lambda_1+\lambda_2+\lambda_3+\lambda_4 = 1, \tag{3.14}$$

and related to Cartesian coordinates via

$$\mathbf{x} = \begin{pmatrix} -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{pmatrix} \boldsymbol{\lambda}, \tag{3.15}$$



(a) Tetrahedron. **(b)** Prism.

(c) Pyramid. **(d)** Hexahedron.

Figure 3.2. Reference domains in three dimensions.

where as with the triangle the columns of the matrix correspond to vertices of the reference tetrahedron. Similarly the symmetric counterparts of $\boldsymbol{\lambda}$ are given by its unique permutations. This leads us to the following five symmetry orbits

$$\begin{aligned}
\mathcal{S}_1 &= (\tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}), & |\mathcal{S}_1| &= 1,\\
\mathcal{S}_2(\alpha) &= \mathrm{Perm}(\alpha, \alpha, \alpha, 1-3\alpha), & |\mathcal{S}_2| &= 4,\\
\mathcal{S}_3(\alpha) &= \mathrm{Perm}(\alpha, \alpha, \tfrac{1}{2}-\alpha, \tfrac{1}{2}-\alpha), & |\mathcal{S}_3| &= 6,\\
\mathcal{S}_4(\alpha,\beta) &= \mathrm{Perm}(\alpha, \alpha, \beta, 1-2\alpha-\beta), & |\mathcal{S}_4| &= 12,\\
\mathcal{S}_5(\alpha,\beta,\gamma) &= \mathrm{Perm}(\alpha, \beta, \gamma, 1-\alpha-\beta-\gamma), & |\mathcal{S}_5| &= 24,
\end{aligned}$$

where α, β, and γ are constrained to ensure that all of the resulting coordinates are inside of the domain.

With some manipulation it can be verified that the orthonormal polynomial basis inside of the reference tetrahedron is given by

$$\psi_{(ijk)}(\mathbf{x}) = \sqrt{8}\hat{P}_i(a)\hat{P}_j^{(2i+1,0)}(b)\hat{P}_k^{(2i+2j+2,0)}(c)(1-b)^i(1-c)^{i+j}, \tag{3.16}$$

where $a = -2(1+x)/(y+z) - 1$, $b = 2(1+y)/(1-z)$, and $c = z$. The objective basis is given by

$$\tilde{\mathcal{P}}^{\phi} = \left\{\psi_{(ijk)}(\mathbf{x}) \mid 0 \le i \le \phi, i \le j \le \phi - i, j \le k \le \phi - i - j\right\}. \tag{3.17}$$

Prism. Extruding the reference triangle along the z-axis gives the reference prism of Figure 3.2. It follows that the volume is $\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{-y} \mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = 4$. There are a total of 12 symmetries. On account of the extrusion the most natural coordinate system is a combination of barycentric and Cartesian coordinates: $(\lambda_1, \lambda_2, \lambda_3, z)$. Let Perm_3 generate all of the unique permutations of its first three arguments. Using this the six symmetry groups of the prism can be expressed as

$$\begin{aligned}
\mathcal{S}_1 &= (\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, 0), & |\mathcal{S}_1| &= 1,\\
\mathcal{S}_2(\gamma) &= (\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \pm\gamma), & |\mathcal{S}_2| &= 2,\\
\mathcal{S}_3(\alpha) &= \mathrm{Perm}_3(\alpha, \alpha, 1-2\alpha, 0), & |\mathcal{S}_3| &= 3,\\
\mathcal{S}_4(\alpha,\gamma) &= \mathrm{Perm}_3(\alpha, \alpha, 1-2\alpha, \pm\gamma), & |\mathcal{S}_4| &= 6,\\
\mathcal{S}_5(\alpha,\beta) &= \mathrm{Perm}_3(\alpha, \beta, 1-\alpha-\beta, 0), & |\mathcal{S}_5| &= 6,\\
\mathcal{S}_6(\alpha,\beta,\gamma) &= \mathrm{Perm}_3(\alpha, \beta, 1-\alpha-\beta, \pm\gamma), & |\mathcal{S}_6| &= 12,
\end{aligned}$$

where the constraints on α and β are identical to those in a triangle and $0 < \gamma \le 1$.

Combining the orthonormal polynomial bases for a right-triangle and line segment yields the orthonormal prism basis

$$\psi_{(ijk)}(\mathbf{x}) = \sqrt{2}\hat{P}_i(a)\hat{P}_j^{(2i+1,0)}(b)\hat{P}_k(c)(1-b)^i, \tag{3.18}$$

where $a = 2(1+x)/(1-y) - 1$, $b = y$, and $c = z$. The objective basis is given by

$$\tilde{\mathcal{P}}^{\phi} = \left\{\psi_{(ijk)}(\mathbf{x}) \mid 0 \le i \le \phi,\ i \le j \le \phi - i,\ 0 \le k \le \phi - i - j,\ k \text{ even}\right\}. \tag{3.19}$$

Pyramid. The reference pyramid can be seen in Figure 3.2 with a volume determined by $\int_{-1}^{1}\int_{(z-1)/2}^{(1-z)/2}\int_{(z-1)/2}^{(1-z)/2} \mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = 8/3$. The symmetries are identical to those of a quadrilateral. Extending the notation employed for the quadrilateral the following symmetry orbits are obtained

$$\begin{aligned}
\mathcal{S}_1(\gamma) &= (0,0,\gamma), & |\mathcal{S}_1| &= 1,\\
\mathcal{S}_2(\alpha,\gamma) &= \chi(\alpha,0,\gamma), & |\mathcal{S}_2| &= 4,\\
\mathcal{S}_3(\alpha,\gamma) &= \chi(\alpha,\alpha,\gamma), & |\mathcal{S}_3| &= 4,\\
\mathcal{S}_4(\alpha,\beta,\gamma) &= \chi(\alpha,\beta,\gamma), & |\mathcal{S}_4| &= 8,
\end{aligned}$$

subject to the constraints that $0 < (\alpha,\beta) \leq (1-\gamma)/2$ and $-1 \leq \gamma \leq 1$.

Inside of the reference pyramid the orthonormal polynomial basis is found to be

$$\psi_{(ijk)}(\mathbf{x}) = 2\hat{P}_i(a)\hat{P}_j(b)\hat{P}_k^{(2i+2j+2,0)}(c)(1-c)^{i+j}, \tag{3.20}$$

where $a = 2x/(1-z)$, $b = 2y/(1-z)$, and $c = z$. The objective basis is

$$\tilde{\mathcal{P}}^{\phi} = \left\{\psi_{(ijk)}(\mathbf{x}) \mid 0 \leq i \leq \phi,\ i \leq j \leq \phi - i,\ 0 \leq k \leq \phi - i - j,\ (i,j)\ \text{even}\right\}. \tag{3.21}$$

Hexahedron. The chosen reference hexahedron can be seen in Figure 3.2. The volume is, trivially, $\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = 8$. A hexahedron exhibits octahedral symmetry with a symmetry number of 48. The procedure for determining the orbits is similar to that used for the quadrilateral. Consider applying these symmetries to a point (α,β,γ) with $0 \leq (\alpha,\beta,\gamma) \leq 1$ and let the resulting set of points by given by $\Xi(\alpha,\beta,\gamma)$. When α, β, and γ are all distinct and greater than zero the set has a cardinality of 48, as expected. However, when one or more parameters are either identical to one another or equal to zero some symmetries give rise to equivalent points. This reduces the cardinality of the

set. Enumerating the various combinations, seven symmetry orbits are obtained

$$\begin{aligned}
\mathcal{S}_1 &= \Xi(0,0,0), & |\mathcal{S}_1| &= 1,\\
\mathcal{S}_2(\alpha) &= \Xi(\alpha,0,0), & |\mathcal{S}_2| &= 6,\\
\mathcal{S}_3(\alpha) &= \Xi(\alpha,\alpha,\alpha), & |\mathcal{S}_3| &= 8,\\
\mathcal{S}_4(\alpha) &= \Xi(\alpha,\alpha,0), & |\mathcal{S}_4| &= 12,\\
\mathcal{S}_5(\alpha,\beta) &= \Xi(\alpha,\beta,0), & |\mathcal{S}_5| &= 24,\\
\mathcal{S}_6(\alpha,\beta) &= \Xi(\alpha,\alpha,\beta), & |\mathcal{S}_6| &= 24,\\
\mathcal{S}_7(\alpha,\beta,\gamma) &= \Xi(\alpha,\beta,\gamma), & |\mathcal{S}_7| &= 48.
\end{aligned}$$

Trivially, the orthonormal basis inside of the reference hexahedron is given by

$$\psi_{(ijk)}(\mathbf{x}) = \hat{P}_i(a)\hat{P}_j(b)\hat{P}_k(c), \tag{3.22}$$

where $a = x$, $b = y$, and $c = z$. The objective basis is

$$\tilde{\mathcal{P}}^{\phi} = \left\{\psi_{(ijk)}(\mathbf{x}) \mid 0 \le i \le \phi,\ i \le j \le \phi - i,\ j \le k \le \phi - i - j,\ (i,j,k) \text{ even}\right\}. \tag{3.23}$$

3.4 Methodology

The methodology employed for identifying symmetric quadrature rules is a refinement of that described by Witherden and Vincent [37] for triangles. This method is, in turn, a refinement of that of Zhang et al. [47].

To derive a quadrature rule, four input parameters are required: the reference domain, $\mathbf{\Omega}$, the number of quadrature points N_p, the target rule strength, ϕ, and a desired runtime, t. The algorithm begins by computing all of the possible symmetric decompositions of N_p. The result is a set of vectors satisfying the relation $\forall i$

$$N_p = \sum_{j=1}^{N_s} n_{ij}|\mathcal{S}_j|, \tag{3.24}$$

where N_s is the number of symmetry orbits associated with the domain $\mathbf{\Omega}$, and n_{ij} is the number of orbits of type j in the ith decomposition. Finding these involves solving the

constrained linear Diophantine equation outlined in §3.3. It is possible for this equation to have no solutions. Consider for example the case of $N_p = 44$ for a triangular domain. From the symmetry orbits

$$N_p = n_1|\mathcal{S}_1| + n_2|\mathcal{S}_2| + n_3|\mathcal{S}_3| = n_1 + 3n_2 + 6n_3,$$

subject to the constraint that $n_1 \in \{0, 1\}$. This restricts N_p to be either a multiple of three or one greater. Since 44 is neither of these the equation is found to have no solutions. Therefore, it is concluded that there can be no symmetric quadrature rules inside of a triangle with 44 points.

Given a decomposition the goal is to find a set of orbital parameters and weights that minimise the error associated with integrating the objective basis on $\mathbf{\Omega}$. This is an example of a non-linear least squares problem. A suitable method for solving such problems is the Levenberg-Marquardt algorithm (LMA). The LMA is an iterative procedure for finding a set of parameters that correspond to a local minima of a set of functions. The minimisation process is not always successful and is dependent on an initial guess of the parameters. Within the context of quadrature rule derivation minimisation can be regarded as successful if $\xi(\phi) \sim \epsilon$ where ϵ represents machine precision.

Let us denote the number of parameters associated with symmetry orbit $\mathcal{S}_i$ as $[\![\mathcal{S}_i]\!]$. Using this the total number of degrees of freedom associated with decomposition i can be expressed as

$$\sum_{j=1}^{N_s} \Big\{ n_{ij} [\![\mathcal{S}_i]\!] + n_{ij} \Big\}, \tag{3.25}$$

with the second term accounting for the presence of one quadrature weight associated with each symmetry orbit. From the list of orbits given in §3.3 the weights are expected to contribute approximately one third of the degrees of freedom. This is not an insignificant fraction. One way of eliminating the weights is to treat them as dependent variables. When the points are prescribed the right hand side of (3.1) becomes linear with respect to the unknowns—the weights. In general, however, the number of weights will be different from the number of polynomials in the objective basis.

It is therefore necessary to obtain a least squares solution to the system. Linear least squares problems can be solved directly through a variety of techniques. Perhaps the most robust numerical scheme is that of singular value decomposition (SVD). Thus, at the cost of solving a small linear least squares problem at each LMA iteration it is possible to reduce the number of free parameters to

$$\sum_{j=1}^{N_s} n_{ij}[\![\mathcal{S}_i]\!]. \tag{3.26}$$

Such a modification has been found to greatly reduce the number of iterations required for convergence. This reduction more than offsets the marginally greater computational cost associated with each iteration.

Previous works [47, 48, 54] have emphasised the importance of picking a 'good' initial guess to seed the LMA. To this end several methodologies for seeding orbital parameters have been proposed. The degree of complexity associated with such strategies is not insignificant. Further, it is necessary to devise a separate strategy for each symmetry orbit. Experience, however, suggests that the choice of decomposition is far more important than the initial guess in determining whether minimisation will be successful. Orbits can therefore be seeded independently using uniform random deviates. Let U_1 be a deviate between $[0, 1]$, U_2 between $[0, 1/2]$, U_3 between $[0, 1/3]$, U_4 between $[0, 1/4]$, U_{11} between $[-1, 1]$ and, U_γ between $[0, (1-\gamma)/2]$. Using these the various orbital parameters can be seeded as follows.

Triangle.

$$\mathcal{S}_2(\alpha \sim U_2)$$
$$\mathcal{S}_3(\alpha \sim U_2, \beta \sim U_3)$$

Quadrilateral.

$$\mathcal{S}_2(\alpha \sim U_1)$$
$$\mathcal{S}_3(\alpha \sim U_1)$$
$$\mathcal{S}_4(\alpha \sim U_1, \beta \sim U_1)$$

Tetrahedron.

$$\mathcal{S}_2(\alpha \sim U_3)$$
$$\mathcal{S}_3(\alpha \sim U_2)$$
$$\mathcal{S}_4(\alpha \sim U_3, \beta \sim U_3)$$
$$\mathcal{S}_5(\alpha \sim U_4, \beta \sim U_4, \gamma \sim U_4)$$

Prism.

$$\mathcal{S}_2(\gamma \sim U_1)$$
$$\mathcal{S}_3(\alpha \sim U_2)$$
$$\mathcal{S}_4(\alpha \sim U_2, \gamma \sim U_1)$$
$$\mathcal{S}_5(\alpha \sim U_2, \beta \sim U_2)$$
$$\mathcal{S}_6(\alpha \sim U_2, \beta \sim U_2, \gamma \sim U_1)$$

Pyramid.

$$\mathcal{S}_1(\gamma \sim U_{11})$$
$$\mathcal{S}_2(\alpha \sim U_\gamma, \gamma \sim U_{11})$$
$$\mathcal{S}_3(\alpha \sim U_\gamma, \gamma \sim U_{11})$$
$$\mathcal{S}_4(\alpha \sim U_\gamma, \beta \sim U_\gamma, \gamma \sim U_{11})$$

Hexahedron.

$$\mathcal{S}_2(\alpha \sim U_1)$$
$$\mathcal{S}_3(\alpha \sim U_1)$$
$$\mathcal{S}_4(\alpha \sim U_1)$$
$$\mathcal{S}_5(\alpha \sim U_1, \beta \sim U_1)$$
$$\mathcal{S}_6(\alpha \sim U_1, \beta \sim U_1)$$
$$\mathcal{S}_7(\alpha \sim U_1, \beta \sim U_1, \gamma \sim U_1)$$

For larger values of N_p it is observed that many decompositions—especially those for prisms and pyramids—are pathological. As an example of this consider searching for an $N_p = 80$ point rule inside of a prism where there are 2 380 distinct symmetrical decompositions. One such decomposition is $N_p = 40|\mathcal{S}_2|$ where all points lie in a single column down the middle of the prism. Since there is no variation in either x or y it is not possible to obtain a rule of strength $\phi \geq 1$. Hence, the decomposition can be dismissed without further consideration.

A presentation of this method in pseudocode can be seen in Algorithm 3.1. When the objective basis functions in $\tilde{\mathcal{P}}^{\phi}$ are orthonormal (3.6) states that the integrand of all non-constant modes is zero. This property can be exploited to greatly simplify the computation of b_i. The purpose of ClampOrbit is to enforce the constraints associated with a given orbit to ensure that all points remain inside of the domain.

3.5 Implementation

The algorithms outlined above have been implemented in a C++11 program called *polyquad*. The program is built on top of the Eigen template library [60] and is parallelised using MPI. It is capable of searching for quadrature rules on triangles, quadrilaterals, tetrahedra, prisms, pyramids, and hexahedra. All rules are guaranteed to be symmetric having all points inside of the domain. Polyquad can also, optionally, filter out rules possessing negative weights. Further, functionality exists, courtesy of MPFR [61], for refining rules to an arbitrary degree of numerical precision and for evaluating the truncation error of a ruleset.

As a point of reference the case of using polyquad to identify $\phi = 10$ rules with 81 points inside of a tetrahedra is considered. Running polyquad for one hour on a quad-core Intel i7-4820K CPU a total of 50 distinct rules were found. These rules were split across four distinct orbital decompositions.

The source code for polyquad is available under the terms of the GNU General Public License v3.0 and can be downloaded from `https://github.com/vincentlab/Polyquad`.

3.6 Rules

Using polyquad a set of quadrature rules for each of the reference domains in §3.3 have been derived. All rules are completely symmetric, possess only positive weights, and have all points inside of the domain. It is customary in the literature to refer to quadratures with the last two attributes as being “PI” rules. As polyquad attempts to

Algorithm 3.1. Procedure for generating symmetric quadrature rules of strength ϕ with N_p points inside of a domain.

```
1:  procedure FindRules(N_p, φ, t)
2:      for all decompositions of N_p do
3:          t_0 ← CurrentTime()
4:          repeat
5:              R ← SeedOrbits()                          ▷ Initial guess of points
6:              ξ ← LMA(RuleResid, R)
7:              if ξ ~ ε then                             ▷ If minimisation was successful
8:                  save R
9:              end if
10:         until CurrentTime() − t_0 > t
11:     end for
12: end procedure

13: function RuleResid(R)
14:     for all p_i ∈ P̃^φ do                             ▷ For each basis function
15:         b_i ← ∫_Ω p_i(x) dx
16:         for all r_j ∈ R do                            ▷ For each orbit
17:             r_j ← ClampOrbit(r_j)                     ▷ Ensure orbital parameters are valid
18:             A_ij ← 0
19:             for all x_k ∈ ExpandOrbit(r_j) do
20:                 A_ij ← A_ij + p_i(x_k)
21:             end for
22:         end for
23:     end for
24:     ω ← b/A                                           ▷ Use SVD to determine the weights
25:     return Aω − b                                     ▷ Compute the residual
26: end function
```

find an ensemble of rules it is necessary to have a means of differentiating between otherwise equivalent formulae. In constructing this collection the truncation term $\xi(\phi + 1)$ was employed with the rule possessing the smallest such term being chosen. The number of points N_p required for a rule of strength ϕ can be seen in Table 3.1. The rules themselves are provided as electronic supplementary material and have been refined to 38 decimal places.

From Table 3.1 it can be seen that several of the rules appear to improve over those in the literature. A rule is considered to be an improvement when it either requires fewer points than any symmetric rule described in literature or when existing symmetric rules of this strength are not PI. For example, many of the rules presented by Dunavant for quadrilaterals [50] and hexahedra [57] possess either negative weights or have points outside of the domain. Using polyquad in quadrilaterals it was possible to identify PI rules with point counts less than or equal to those of Dunavant at strengths $\phi = 8, 9, 18, 19, 20$. In tetrahedra it was possible to reduce the number of points required for $\phi = 7$ and $\phi = 9$ by one and two, respectively, compared with Zhang et al. [47]. Furthermore, in prisms and pyramids rules requiring significantly fewer points than those in literature were identified. As an example, the $\phi = 9$ rule of [54] inside of a prism requires 71 points compared with just 60 for the rule identified by polyquad. Additionally, both of the $\phi = 10$ rules for prisms and pyramids appear to be new.

Table 3.1. Number of points N_p required for a fully symmetric quadrature rule with positive weights of strength ϕ. Rules with underlines represent improvements over those found in the literature (see text).

	N_p					
ϕ	Tri	Quad	Tet	Pri	Pyr	Hex
1	1	1	1	1	1	1
2	3	4	4	5	5	6
3	6	4	8	8	6	6
4	6	8	14	11	10	14
5	7	8	14	16	15	14
6	12	12	24	28	24	34
7	15	12	35	35	31	34
8	16	20	46	46	47	58
9	19	20	59	60	62	58
10	25	28	81	85	83	90
11	28	28				
12	33	37				
13	37	37				
14	42	48				
15	49	48				
16	55	60				
17	60	60				
18	67	72				
19	73	72				
20	79	85				

Chapter 4

Solution and Flux Points

4.1 Requirements

From the one dimensional numerical experiments of §2.4 it is apparent that the choice of solution/flux points can have a significant impact on the quality of the resulting interpolating polynomial. The first question that must be considered, however, is that of existence: given a set of points inside of an element $\{\tilde{\mathbf{x}}_{e\rho}\}$ is it possible to construct a nodal basis set?

In one dimension, closed form expressions exist for the nodal basis set with

$$\ell_\rho(\tilde{x}) = \prod_{\sigma \neq \rho} \frac{\tilde{x} - \tilde{x}_\sigma}{\tilde{x}_\rho - \tilde{x}_\sigma}, \tag{4.1}$$

where it can be readily verified that $\ell_\rho(\tilde{x}_\sigma) = \delta_{\rho\sigma}$. By inspection it is clear that the only requirement on the points is that they all be distinct. Beyond one dimension the set of nodal basis functions is defined through the inverse of the generalised Vandermonde matrix as

$$\ell_{e\rho}(\tilde{\mathbf{x}}) = \mathcal{V}^{-1}_{e\rho\sigma} \psi_{e\sigma}(\tilde{\mathbf{x}}), \tag{4.2}$$

where $\mathcal{V}_{e\rho\sigma} = \psi_{e\rho}(\tilde{\mathbf{x}}_{e\sigma})$. To be able to build the nodal basis set it is therefore necessary for $\mathcal{V}_e$ to be invertible. This is equivalent to requiring that $\det \mathcal{V}_e \neq 0$. It is known that in two and three dimensions that only certain sets of points fulfil this requirement which is termed *unisolvency* [6, 62]. To illustrate this take $\{\tilde{\mathbf{x}}_{e\rho}\}$ to be a set of distinct points with a non-singular Vandermonde matrix and consider arbitrarily relabelling a pair of points. The effect of this relabelling is to interchange two columns in the Vandermonde matrix. From the properties of the determinant this will cause the sign of the determinant to flip. If this interchange is performed continuously, with the two points following di fferent

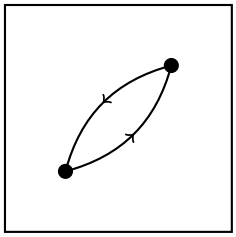


Figure 4.1. Origins of non-unisolvency.

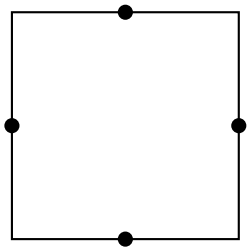


Figure 4.2. A set points that are not unisolvent.

non-intersecting paths as shown in Figure 4.1, it is evident from the intermediate value theorem that there must be an intermediate arrangement where the determinant is zero. Hence, while the points are all distinct they can not be used to construct a nodal basis set.

The next requirement is that of symmetry. This is essential for flux points as otherwise there can exist topological configurations where pairs of flux points do not align in physical space. Although there is no formal requirement for the solution points to be symmetric it is nevertheless desirable for them to respect the underlying symmetries of the element.

The quality of the interpolating polynomial $p(\mathbf{x})$ arising from a collocation projection of a function $f(\mathbf{x})$ can be measured by taking a norm of $f(\mathbf{x}) - p(\mathbf{x})$ over the region of interest. Traditionally, the majority of nodal finite element codes have eschewed collocation type projections in lieu of full L^2 projections using quadrature. Within this framework the role of solution points is reduced to that of polynomial interpolation. Hence, the main criterion used to assess the suitability of a set of points is the conditioning of the resulting nodal basis set. This property is most naturally assessed by considering the L^∞ norm. Minimising this yields the so-called minmax polynomial; that with the smallest maximum deviation. Denoting this polynomial as $p^\star(\mathbf{x})$ and

following the approach of Rivlin [63] it is observed that

$$\begin{aligned}
\|f(\mathbf{x}) - p(\mathbf{x})\|_\infty &= \|f(\mathbf{x}) - p^\star(\mathbf{x}) + p^\star(\mathbf{x}) - p(\mathbf{x})\|_\infty \\
&\leq \|f(\mathbf{x}) - p^\star(\mathbf{x})\|_\infty + \|p^\star(\mathbf{x}) - p(\mathbf{x})\|_\infty \\
&= \|f(\mathbf{x}) - p^\star(\mathbf{x})\|_\infty + \left\| \sum_i [p^\star(\mathbf{x}_i) - f(\mathbf{x}_i)]\ell_i(\mathbf{x}) \right\|_\infty \\
&\leq \|f(\mathbf{x}) - p^\star(\mathbf{x})\|_\infty + \max_{\mathbf{x}\in\mathbf{\Omega}} |p^\star(\mathbf{x}) - f(\mathbf{x})| \cdot \left\| \sum_i \ell_i(\mathbf{x}) \right\|_\infty \\
&\leq \|f(\mathbf{x}) - p^\star(\mathbf{x})\|_\infty + \max_{\mathbf{x}\in\mathbf{\Omega}} |p^\star(\mathbf{x}) - f(\mathbf{x})| \cdot \max_{\mathbf{x}\in\mathbf{\Omega}} \sum_i |\ell_i(\mathbf{x})| \\
&= \|f(\mathbf{x}) - p^\star(\mathbf{x})\|_\infty + \|f(\mathbf{x}) - p^\star(\mathbf{x})\|_\infty \cdot \max_{\mathbf{x}\in\mathbf{\Omega}} \sum_i |\ell_i(\mathbf{x})| \\
&= (1 + \Lambda)\|f(\mathbf{x}) - p^\star(\mathbf{x})\|_\infty,
\end{aligned} \tag{4.3}$$

where

$$\Lambda = \max_{\mathbf{x}\in\mathbf{\Omega}} \sum_i |\ell_i(\mathbf{x})|, \tag{4.4}$$

is known as the Lebesgue constant. Unsurprisingly, there is an extensive body of literature regarding the generation of point sets which minimise Λ in triangles [5, 11, 62, 64, 65], tetrahedra [11, 66, 67], and most recently pyramids [68].

When collocation projections are employed, however, the solution points play a dual role [33, 37]. In addition to polynomial interpolation they are also used to project potentially non-linear functions into the polynomial space of the solution. Results from §2.4 suggest that in these circumstances, the most sensible metric for assessing the suitability of a set of points is the L^2 norm. While using the FR approach to solve the Euler equations on triangular elements Catonguay et al. [34] found that the α-optimised points of Hesthaven and Warburton [11], which are constructed to minimise Λ, exhibited extremely poor performance. These issues were resolved when the points were exchanged for the abscissa of the mildly asymmetric quadrature rules of Taylor et al. [48]. Following on from this Williams et al. [36] proceeded to identify a complete set of fully symmetric quadrature rules in triangles that are suitable for use as solution points.

4.2 Line Segments

In one dimension there exists a closed-form expression for the truncation error associated with a Lagrange type interpolating polynomial

$$p(x) = \sum_i f(x_i)\ell_i(x) + \xi(x), \tag{4.5}$$

where

$$\xi(x) = \frac{f^{(n+1)}(c)}{(n+1)!}\prod_i (x - x_i), \tag{4.6}$$

where c is an unknown constant and n is the number of nodal basis functions. Using this definition a least squares error can be introduced over the domain Ω as

$$\sigma^2 = \int_{-1}^{1} \xi^2(x)\,\mathrm{d}x = A^2 \int_{-1}^{1} \prod_i (x - x_i)^2\,\mathrm{d}x, \tag{4.7}$$

where $A = f^{(n+1)}(c)/(n+1)!$. Under the assumption that A has no dependence on the choice of nodes this can be minimised as

$$\partial_{x_k}\sigma^2 = -2\int_{-1}^{1} \underbrace{\prod_i (x - x_i)}_{\text{degree } n} \underbrace{\prod_{j\neq k} (x - x_j)}_{\text{degree } n-1}\,\mathrm{d}x = 0. \tag{4.8}$$

This equation can be solved by requiring that the first term inside of the integral, of degree n, be orthogonal to all polynomials of degree $n-1$. The simplest means of satisfying this requirement is to let the first term be a Legendre polynomial of degree n

$$P_n(x) = \prod_i (x - x_i), \tag{4.9}$$

with the solution points being given as the roots of $P_n(x)$. This is the very definition of the abscissa of the n point Gauss-Legendre quadrature rule. Hence, in the absence of any additional information about the form of $f(x)$, it has been shown that when performing a collocation projection that the Gauss-Legendre points are optimal in the least squares sense. This result is in good agreement with those of §2.4 where in

the one dimensional numerical experiments the Gauss-Legendre points were found to outperform the Gauss-Legendre-Lobatto and equispaced points. Further, it is also consistent with the theoretical arguments of Jameson et al. [33].

Through a tensor product construction this result can be extended to both quadrilaterals and hexahedra. As the nodal basis functions inside these domains are simply a product of one dimensional Lagrange nodal basis functions the unisolvency of the basis follows immediately from the uniqueness of the Gauss-Legendre points. However, from chapter 3 it is known there exist symmetric arrangements of points inside of quadrilaterals and hexahedra that do not correspond to any tensor product construction. Thus, the resulting point sets can not be considered globally optimal.

4.3 Triangles

Beyond tensor product elements it is not possible to obtain an analytic expression for the truncation error. This precludes any direct numerical analysis or optimisation. However, from the previous work of Catonguay et al. [34] and Williams at al. [35] there is strong empirical evidence to suggest that solution points should be placed at the abscissa of strong Gaussian quadrature rules. In this section further consideration is given to this notion by analysing the impact of solution point placement inside of triangles [37].

Candidate point sets. Using a precursor to Polyquad a large number of symmetric quadrature rules with a triangular number of points were generated. A summary of the rules found can be seen in Table 4.1. The quadrature rule strengths were selected to be the strongest obtainable for the specified number of points. However, at polynomial orders five and six many/all of the highest strength rules that were found yielded singular Vandermonde matrices. Hence, in both instances the derivation procedure was repeated with a lower target rule strength. Truncation errors were computed in accordance with (3.7).

To facilitate a comparison of these points against those in the literature the α-optimised points of Hesthaven and Warburton, which can be viewed as a generalisation

Table 4.1. Number of rules N_r discovered at each polynomial order $\wp$ and the associated quadrature strengths ϕ. The basis order used for computing the truncation error is indicated by ϕ^+.

$\wp$	3	4	5	5	6	6	7
ϕ	5	7	8	9	10	11	12
ϕ^+	6	9	10	10	12	12	14
N_r	95	66	722	473	412	12	136
$N_r(\det \mathcal{V} \neq 0)$	24	64	452	2	236	0	100

of the Gauss-Legendre-Lobatto points and are designed to minimise Λ, were selected along with the symmetric quadrature rules of Williams et al. [36] which herein shall be referred to as the WS points.

Numerical experiments. Following [23, 34, 35] the numerical performance of these solution points shall be evaluated by simulating an isentropic Euler vortex in a free-stream. The initial conditions for this numerical experiment are given by

$$\rho(\mathbf{x}, t=0) = \left\{1 - \frac{S^2 M^2 (\gamma-1)\exp 2f}{8\pi^2}\right\}^{\frac{1}{\gamma-1}}, \tag{4.10}$$

$$\mathbf{v}(\mathbf{x}, t=0) = \frac{S\, y \exp f}{2\pi R}\hat{\mathbf{x}} + \left\{1 - \frac{S\, x \exp f}{2\pi R}\right\}\hat{\mathbf{y}}, \tag{4.11}$$

$$p(\mathbf{x}, t=0) = \frac{\rho^\gamma}{\gamma M^2}, \tag{4.12}$$

where $f = (1 - x^2 - y^2)/2R^2$, $S = 13.5$ is the strength of the vortex, $M = 0.4$ is the free-stream Mach number, and $R = 1.5$ is the radius. The domain was taken to be $\mathbf{\Omega} = [-10, 10]^2$. All meshes were configured with fully periodic boundary conditions. These conditions, however, result in the modelling of an infinite grid of coupled vortices [23]. The impact of this is mitigated by the observation that the exponentially-decaying vortex has a radius which is far smaller than the extent of the domain. Neglecting these

effects the analytic solution of the system at a time t is simply a translation of the initial conditions.

Using the analytical solution an L^2 error can be defined as

$$\sigma(t)^2 = \int_{-2}^{2}\int_{-2}^{2} \left[\rho^{\delta}(\mathbf{x} + \Delta_y(t)\hat{\mathbf{y}}, t) - \rho(\mathbf{x} + \Delta_y(t)\hat{\mathbf{y}}, t)\right]^2 \mathrm{d}^2\mathbf{x}, \tag{4.13}$$

where $\rho^{\delta}(\mathbf{x}, t)$ is the numerical mass density, $\rho(\mathbf{x}, t)$ the analytic mass density, and $\Delta_y(t)$ is the ordinate corresponding to the centre of the vortex at t and accounts for the fact that the vortex is translating in a free stream of velocity unity in the y direction. Restricting the region of consideration to a small box centred around the vortex serves to further mitigate against the effects of vortices coupling together. The initial mass density along with the $[-2, -2] \times [2, 2]$ region used to evaluate the error can be seen in Figure 4.3. For an arbitrary triangular mesh the evaluation of (4.13) is somewhat cumbersome. However, if the mesh is constructed such that there are times, t_c, when the region of integration does not straddle any mesh elements then the error can be computed by simply integrating over each element and summing the residuals

$$\begin{aligned}\sigma(t_c)^2 &= \iint_{\hat{\mathbf{\Omega}}_e} \left[\rho_i^{\delta}(\tilde{\mathbf{x}}, t_c) - \rho(\boldsymbol{\mathcal{M}}_i(\tilde{\mathbf{x}}), t_c)\right]^2 J_i(\tilde{\mathbf{x}})\, \mathrm{d}^2\tilde{\mathbf{x}} \\ &\approx \left[\rho_i^{\delta}(\tilde{\mathbf{x}}_j, t_c) - \rho(\boldsymbol{\mathcal{M}}_i(\tilde{\mathbf{x}}_j), t_c)\right]^2 J_i(\tilde{\mathbf{x}}_j)\omega_j,\end{aligned} \tag{4.14}$$

where, $\rho_i^{\delta}(\tilde{\mathbf{x}}, t_c)$ is the approximate mass density inside of the ith element in the box, and $J_i(\tilde{\mathbf{x}})$ the associated Jacobian. In the second step the integral has been approximated using a quadrature rule with abscissa $\{\tilde{\mathbf{x}}_j\}$ and weights $\{\omega_j\}$. The requirement that there exist times when the grid and bounding box conform has been satisfied by using the mesh of Figure 4.4.

To completely specify the proposed numerical experiment it is also necessary to specify the time-marching algorithm/time-step, the approximate Riemann solver, and the choice of flux points along each edge. In this study a classical fourth-order Runge–Kutta (RK4) scheme is chosen with $\Delta t = 0.0005$. For computing the numerical fluxes at element interfaces a Rusanov type Riemann solver, as presented in [9] and detailed in §A.1, is employed. Finally, at the edges of the triangles, the flux points are taken to be at Gauss-Legendre points.

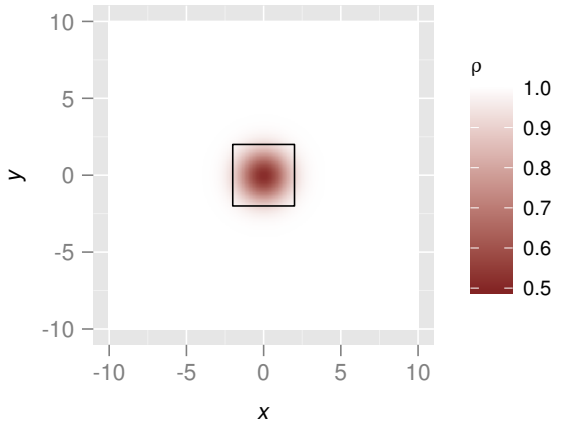


Figure 4.3. Initial density profile for the vortex in Ω. The black box shows the area where the error is calculated at $t = 0$. This box remains centred on the vortex as it progress in the $+y$ direction.

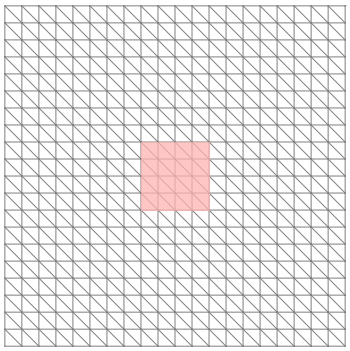
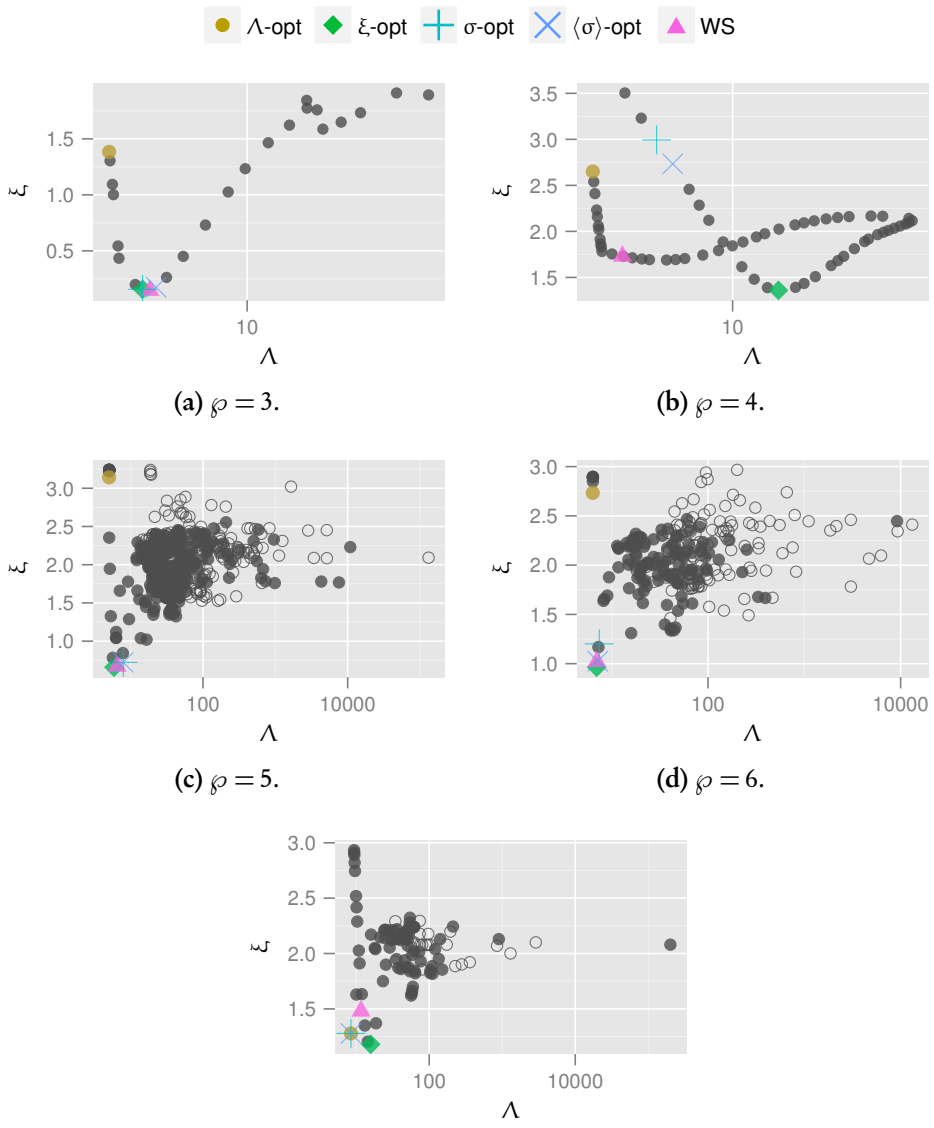


Figure 4.4. The mesh used for the vortex test case consisting of 800 triangles.

Results and discussion. For each order, all derived point sets were subjected to the Euler vortex test case. Simulations were run until $t = 100$; equivalent to five passes of the vortex through the domain. Measurements of the error were made every time unit with the simulation being terminated should NaNs be encountered. For each rule there are three direct metrics: the Lebesgue constant, Λ, the truncation error, ξ, and the binary measure of whether the simulation made it to $t = 100$ or not. Further, for those rules where the vortex does not break down it is possible to compute the L^2 error at $t = 100$, σ, and the average L^2 error over the entire simulation, $\langle\sigma\rangle$. Denote the rule with the smallest L^2 error at $t = 100$ as being the σ-optimal point set and the rule with the smallest average L^2 error as being the $\langle\sigma\rangle$-optimal set. The range of Lebesgue constants and truncation errors at each order can be seen in Figure 4.5. Plots of the L^2 error against time are shown in Figure 4.6.

Starting with the Lebesgue-truncation plots, it is evident that for all orders except $\wp = 4$ the σ- and $\langle\sigma\rangle$-optimal point sets—along with those of Williams et al. [36]—feature both low Lebesgue constants and low truncation errors. At higher orders it is evident that points with either high Lebesgue constants or high truncation errors are more likely to either become unstable before $t = 100$ or perform poorly. A good example of this is the Λ-optimal points at orders $\wp = 3, 5, 6$. At these orders the Λ-optimal points all have truncation errors within the upper-quartile and exhibit markedly worse performance than the σ-optimal or WS points. Conversely, at $\wp = 7$ when the Λ-optimal point set has a truncation error which lies in the lower-quartile of the distribution the performance of the set is extremely good. These three results all serve to reaffirm the dual role that solution points play in FR schemes.

From the error-time plots it is observed that for all polynomial orders the performance of the α-optimised points is significantly worse than those which are good quadrature rules. This is in good agreement with Castonguay et al. [34]. It is also observed from Table 4.2 that at orders $\wp = 4, 6, 7$ the σ-optimal rule sets outperform the WS points by 73%, 33%, and 34% respectively.

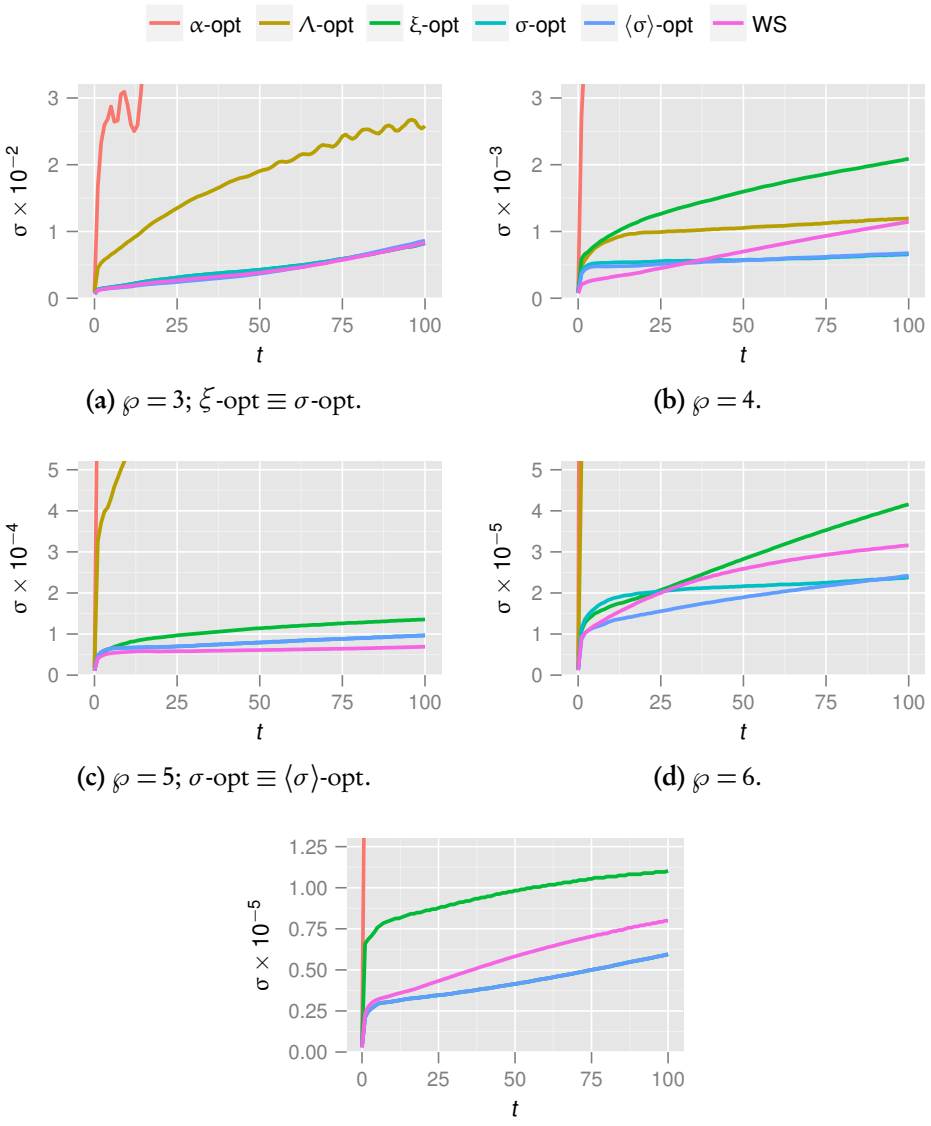


(a) $\wp = 3$. (b) $\wp = 4$.

(c) $\wp = 5$. (d) $\wp = 6$.

(e) $\wp = 7$.

Figure 4.5. Semi-log plots of the Lebesgue constant Λ against truncation error ξ for all point sets. Rules which do not make it to $t = 100$ are indicated by hollow markers.



(a) $\wp = 3$; ξ-opt $\equiv$ σ-opt.

(b) $\wp = 4$.

(c) $\wp = 5$; σ-opt $\equiv$ $\langle\sigma\rangle$-opt.

(d) $\wp = 6$.

(e) $\wp = 7$; Λ-opt $\equiv$ σ-opt $\equiv$ $\langle\sigma\rangle$-opt.

Figure 4.6. L^2 error against time for the α-optimised, Λ-optimal, ξ-optimal, σ-optimal, $\langle\sigma\rangle$-optimal, and WS points.

Table 4.2. L^2 errors at $t = 100$ for the various point sets.

	$\sigma(t = 100)$				
Points	$\wp = 3$	$\wp = 4$	$\wp = 5$	$\wp = 6$	$\wp = 7$
Λ-opt	2.58×10^{-2}	1.20×10^{-3}	2.64×10^{-3}	2.02×10^{-4}	5.95×10^{-6}
ξ-opt	8.20×10^{-3}	2.09×10^{-3}	1.36×10^{-4}	4.16×10^{-5}	1.10×10^{-5}
σ-opt	8.20×10^{-3}	6.59×10^{-4}	9.69×10^{-5}	2.38×10^{-5}	5.95×10^{-5}
$\langle\sigma\rangle$-opt	8.61×10^{-3}	6.76×10^{-4}	9.96×10^{-5}	2.42×10^{-5}	5.95×10^{-6}
WS	8.27×10^{-3}	1.15×10^{-3}	6.92×10^{-5}	3.16×10^{-5}	8.00×10^{-6}

Chapter 5

Implementation

The implementation of the FR approach presented in this thesis is called PyFR. Written in Python, PyFR is designed to be compact, efficient, scalable, and performance portable across a range of platforms. Key functionality is summarised in Table 5.1.

As outlined in §2.5 the majority of operations within an FR step can be cast in terms of matrix-matrix multiplications between a constant operator matrix and a state matrix. All remaining operations, e.g. flux evaluations, are pointwise and concern themselves with either a single solution point inside of an element or two collocating flux points at an interface. Hence, in broad terms there are five salient aspects of an FR implementation, (i) definition of the constant operator matrices, (ii) specification of the state matrices, (iii) implementation of the matrix multiplication kernels, (iv) implementation of the pointwise kernels and, finally (v) handling of distributed memory parallelism and scheduling.

5.1 Definition of Operator Matrices

Setup of the seven constant operator matrices detailed in §2.5 requires evaluation of various polynomial expressions, and their derivatives, at solution/flux points within each type of standard element. Efficiency of this setup phase is not crucial, since the operations are only performed once at start-up. The matrix elements are therefore evaluated using pure Python code via the arbitrary precision mpmath [69] library.

Table 5.1. Key functionality of PyFR v1.0.0.

Dimensions	2D, 3D
Elements	Triangles, Quadrilaterals, Hexahedra, Tetrahedra, Prisms, Pyramids
Spatial orders	Arbitrary
Time steppers	Euler, RK4, RK45
Precisions	Single, Double
Backends	C/OpenMP, CUDA, OpenCL
Communication	MPI
File format	Parallel HDF5
Governing systems	Euler, compressible Navier–Stokes

5.2 Specification of State Matrices

As illustrated by Figure 2.3 there are a variety of approaches for arranging data in memory. For simplicity PyFR utilises the SoA layout. One limitation of this approach is that the stride between consecutive field variables is given by $k = |\mathbf{\Omega}_e|$ and hence is a function of the element type. At element interfaces it is therefore necessary to store both a pointer to the first field variable and the stride. This results in a slight increase in memory usage compared with the AoS and AoSoA approaches where the stride remains identical across all element types. Nevertheless, this limitation is more than offset by, firstly, the desirable coalesced memory access patterns that result from the SoA ordering *cf.* AoS and, secondly, the ease at which these kernels can be automatically vectorised *cf.* AoSoA.

5.3 Matrix Multiplication Kernels

PyFR defers matrix multiplication to the GEMM family of sub-routines provided by a suitable Basic Linear Algebra Subprograms (BLAS) library. BLAS is available for virtually all platforms and optimised versions are often maintained by the hardware

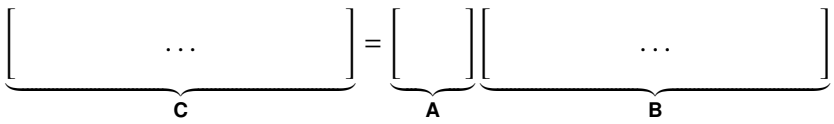


Figure 5.1. Block-by-panel type matrix multiplications for $\mathbf{C} = \mathbf{AB}$ where $\mathbf{A}$ is a constant operator matrix.

vendors themselves, e.g. cuBLAS for NVIDIA GPUs. This approach greatly facilitates development of efficient and platform portable code. However, it is important to note that the matrix sizes encountered in PyFR are not necessarily optimal from a GEMM perspective. Specifically, GEMM is optimised for the multiplication of large square matrices, whereas the constant operator matrices in PyFR are of the block-by-panel variety as illustrated in Figure 5.1. Moreover, the constant operator matrices are know *a priori*, and do not change in time. This knowledge could, in theory, be leveraged to design bespoke matrix multiply kernels that are more efficient than GEMM. Development of such bespoke kernels will be a topic of future research.

5.4 Pointwise Kernels

Pointwise kernels are specified using a domain specific language implemented in PyFR atop of the Mako templating engine [70]. The templated kernels are then interpreted at runtime, converted to low-level code, compiled, linked and loaded. Currently the templating engine can generate C/OpenMP to target CPUs, CUDA via the PyCUDA wrapper [71] to target NVIDIA GPUs, and OpenCL via the PyOpenCL wrapper [71] to target any platform with an OpenCL implementation. Use of a domain specific language avoids implementation of each pointwise kernel for each target platform; keeping the codebase compact and platform portable. Runtime code generation also means it is possible to instruct the compiler to emit binaries which are optimised for the current hardware architecture. Such optimisations can result in substantial improvement in performance when compared with architectural defaults.

```
# -*- coding: utf-8 -*-
<%inherit file='base'/>
<%namespace module='pyfr.backends.base.makoutil' name='pyfr'/>

<%pyfr:kernel name='negdivconf' ndim='2'
              t='scalar fpdtype_t'
              tdivtconf='inout fpdtype_t[${str(nvars)}]'
              ploc='in fpdtype_t[${str(ndims)}]'
              rcpdjac='in fpdtype_t'>
% for i, ex in enumerate(srcex):
    tdivtconf[${i}] = -rcpdjac*tdivtconf[${i}] + ${ex};
% endfor
</%pyfr:kernel>
```

Figure 5.2. PyFR/Mako source for the `negdivconf` kernel.

As an example of a pointwise kernel consider the final evaluation of the the semi-discretised form of the system being solved

$$\frac{\partial u^{(u)}_{e\rho n\alpha}}{\partial t} = -J^{-1\,(u)}_{e\rho n}(\tilde{\boldsymbol{\nabla}} \cdot \tilde{\mathbf{f}})^{(u)}_{e\rho n\alpha} + S^{(u)}_{e\rho n\alpha},$$

where $S^{(u)}_{e\rho n\alpha}$ is a source term that is permitted to vary in both space and time. Figure 5.2 shows how such a kernel can be expressed in the domain specific language of PyFR. There are several points of note. Firstly, the kernel is purely scalar in nature; choices such as how to vectorise a given operation or how to gather data from memory are all delegated to the backend-specific templating engine. Secondly, it is possible to utilise Python when generating the main body of kernels. This capability is used to loop over each of the field variables to generate the body of the kernel. Since kernels are generated at runtime it is trivial to support user-defined source terms. Expressions may be written in the input configuration file and, after some validation, are substituted directly into the kernel as it is being generated. The resulting kernels in the case of$N_V = 4$ with no source terms for the C/OpenMP, CUDA, and OpenCL backends can be seen in Figures 5.3, 5.4, and 5.5 respectively. Observe here the somewhat unconventional structure of the C/OpenMP kernel which is markedly different from the CUDA and OpenCL kernels. This structure is necessary to ensure that the kernel is properly vectorised across a range of compilers.

```c
#define PYFR_ALIGN_BYTES 32
#define PYFR_NOINLINE __attribute__ ((noinline))

typedef double fpdtype_t;

// loop_sched_2d definition omitted

static PYFR_NOINLINE void
negdivconf_inner(int _nx,
                 const fpdtype_t *__restrict__ rcpdjac_v,
                 fpdtype_t *__restrict__ tdivtconf_v0,
                 fpdtype_t *__restrict__ tdivtconf_v1,
                 fpdtype_t *__restrict__ tdivtconf_v2,
                 fpdtype_t *__restrict__ tdivtconf_v3)
{
    for (int _x = 0; _x < _nx; _x++)
    {
        tdivtconf_v0[_x] = -rcpdjac_v[_x]*tdivtconf_v0[_x] + 0;
        tdivtconf_v1[_x] = -rcpdjac_v[_x]*tdivtconf_v1[_x] + 0;
        tdivtconf_v2[_x] = -rcpdjac_v[_x]*tdivtconf_v2[_x] + 0;
        tdivtconf_v3[_x] = -rcpdjac_v[_x]*tdivtconf_v3[_x] + 0;
    }
}

void
negdivconf(int _ny, int _nx,
           const fpdtype_t* __restrict__ rcpdjac_v, int lsdrcpdjac,
           fpdtype_t* __restrict__ tdivtconf_v, int lsdtdivtconf)
{
    #pragma omp parallel
    {
        int align = PYFR_ALIGN_BYTES / sizeof(fpdtype_t);
        int rb, re, cb, ce;
        loop_sched_2d(_ny, _nx, align, &rb, &re, &cb, &ce);

        for (int _y = rb; _y < re; _y++)
            negdivconf_inner(ce - cb,
                             rcpdjac_v + _y*lsdrcpdjac + cb,
                             tdivtconf_v + (_y*4 + 0)*lsdtdivtconf + cb,
                             tdivtconf_v + (_y*4 + 1)*lsdtdivtconf + cb,
                             tdivtconf_v + (_y*4 + 2)*lsdtdivtconf + cb,
                             tdivtconf_v + (_y*4 + 3)*lsdtdivtconf + cb);
    }
}
```

Figure 5.3. Generated OpenMP annotated C code for the `negdivconf` kernel.

```
typedef double fpdtype_t;

__global__ void
negdivconf(int _ny, int _nx,
           const fpdtype_t* __restrict__ rcpdjac_v, int lsdrcpdjac,
           fpdtype_t* __restrict__ tdivtconf_v, int lsdtdivtconf)
{
    int _x = blockIdx.x*blockDim.x + threadIdx.x;

    for (int _y = 0; _y < _ny && _x < _nx; ++_y)
    {
        tdivtconf_v[(_y*4 + 0)*lsdtdivtconf + _x] =
            -rcpdjac_v[lsdrcpdjac*_y + _x]*tdivtconf_v[(_y*4 + 0)*lsdtdivtconf + _x] + 0;
        tdivtconf_v[(_y*4 + 1)*lsdtdivtconf + _x] =
            -rcpdjac_v[lsdrcpdjac*_y + _x]*tdivtconf_v[(_y*4 + 1)*lsdtdivtconf + _x] + 0;
        tdivtconf_v[(_y*4 + 2)*lsdtdivtconf + _x] =
            -rcpdjac_v[lsdrcpdjac*_y + _x]*tdivtconf_v[(_y*4 + 2)*lsdtdivtconf + _x] + 0;
        tdivtconf_v[(_y*4 + 3)*lsdtdivtconf + _x] =
            -rcpdjac_v[lsdrcpdjac*_y + _x]*tdivtconf_v[(_y*4 + 3)*lsdtdivtconf + _x] + 0;
    }
}
```

Figure 5.4. Generated CUDA code for the `negdivconf` kernel.

```
#if __OPENCL_VERSION__ < 120
# pragma OPENCL EXTENSION cl_khr_fp64: enable
#endif

typedef double fpdtype_t;

__kernel void
negdivconf(int _ny, int _nx,
           __global const fpdtype_t* restrict rcpdjac_v, int lsdrcpdjac,
           __global fpdtype_t* restrict tdivtconf_v, int lsdtdivtconf)
{
    int _x = get_global_id(0);

    for (int _y = 0; _y < _ny && _x < _nx; ++_y)
    {
        tdivtconf_v[(_y*4 + 0)*lsdtdivtconf + _x] =
            -rcpdjac_v[lsdrcpdjac*_y + _x]*tdivtconf_v[(_y*4 + 0)*lsdtdivtconf + _x] + 0;
        tdivtconf_v[(_y*4 + 1)*lsdtdivtconf + _x] =
            -rcpdjac_v[lsdrcpdjac*_y + _x]*tdivtconf_v[(_y*4 + 1)*lsdtdivtconf + _x] + 0;
        tdivtconf_v[(_y*4 + 2)*lsdtdivtconf + _x] =
            -rcpdjac_v[lsdrcpdjac*_y + _x]*tdivtconf_v[(_y*4 + 2)*lsdtdivtconf + _x] + 0;
        tdivtconf_v[(_y*4 + 3)*lsdtdivtconf + _x] =
            -rcpdjac_v[lsdrcpdjac*_y + _x]*tdivtconf_v[(_y*4 + 3)*lsdtdivtconf + _x] + 0;
    }
}
```

Figure 5.5. Generated OpenCL code for the `negdivconf` kernel.

5.5 Distributed Memory Parallelism

PyFR is capable of operating on high performance computing clusters utilising distributed memory parallelism. This is accomplished through the Message Passing Interface (MPI). All MPI functionality is implemented at the Python level through the mpi4py [72] wrapper. Parallel I/O is achieved through use of the h5py [73] wrappers around HDF5. To enhance the scalability of the code care has been taken to ensure that all requests are persistent, point-to-point and non-blocking. Further, the format of data that is shared between ranks has been made backend independent. It is therefore possible to deploy PyFR on heterogeneous clusters consisting of both conventional CPUs and accelerators.

The arrangement of kernel calls required to solve an advection-di ffusion problem without anti-aliasing or boundary conditions can be seen in Figure 5.6. The primary objective when scheduling kernels is to maximise the potential for overlapping communication with computation. In order to help achieve this the common interface solution $\mathfrak{C}_\alpha$ and common interface flux $\mathfrak{F}_\alpha$ kernels have been broken apart into two separate kernels; suffixed in the figure by INT and MPI. PyFR is therefore able to perform a significant degree of rank-local computation while the relevant ghost states are being exchanged.

A secondary objective when scheduling kernels is to minimise the amount of temporary storage required. Such optimisations are critical within the context of accelerators which often have an order of magnitude less memory than a contemporary platform. In order to help achieve this $\mathbf{U}^{(u)}$, $\tilde{\mathbf{R}}^{(u)}$, and $\mathbf{R}^{(u)}$ are allowed to alias. By permitting the same storage location to be used for both the inputted solution and the outputted flux divergence it is possible to reduce the storage requirements of the RK schemes. Another opportunity for memory reuse is in the transformed flux function where the incoming gradients $\mathbf{Q}^{(u)}$ can be overwritten with the transformed flux $\tilde{\mathbf{F}}^{(u)}$. A similar approach can be used in the common interface flux function whereby $\mathbf{U}^{(f)}$ can updated in-place with the entires of $\tilde{\mathbf{D}}^{(f)}$ which holds the transformed common normal flux. Moreover, $\mathbf{C}^{(f)}$ is also able to utilise the same storage as the somewhat larger $\mathbf{Q}^{(f)}$ array.

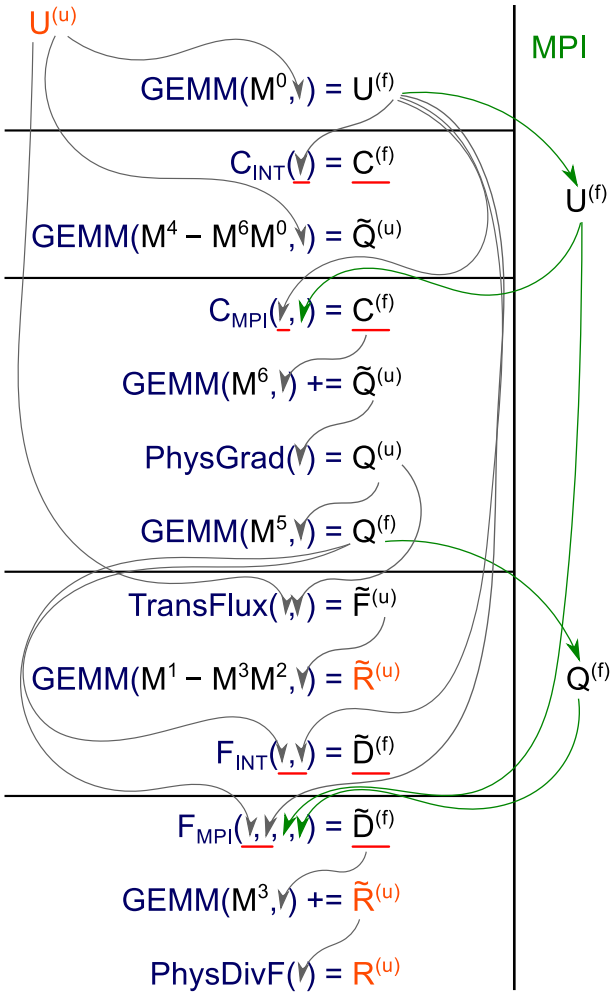


Figure 5.6. Flow diagram showing the stages required to compute $\nabla \cdot \boldsymbol{f}$. Symbols correspond to those of §2.5. For simplicity arguments referencing constant data have been omitted. Memory indirection is indicated by red underlines. Synchronisation points are signified by black horizontal lines.

Chapter 6

Validation

6.1 Euler Vortex

Various authors [10, 23] have shown FR schemes exhibit so-called 'super accuracy' where the order of accuracy is greater than the expected $\wp + 1$. To confirm that PyFR can achieve super accuracy for the Euler equations the isentropic Euler vortex test case of §4.3 was employed. As a means of further mitigation against the effect of interacting vortices the size of the domain was increased to $\mathbf{\Omega} = [-20, 20]^2$. Additionally, along boundaries of constant y the dynamical variables were fixed according to

$$\begin{aligned}
\rho(\mathbf{x} = x\hat{\mathbf{x}} \pm 20\hat{\mathbf{y}}, t) &= 1, \\
\mathbf{v}(\mathbf{x} = x\hat{\mathbf{x}} \pm 20\hat{\mathbf{y}}, t) &= \hat{\mathbf{y}}, \\
p(\mathbf{x} = x\hat{\mathbf{x}} \pm 20\hat{\mathbf{y}}, t) &= \frac{1}{\gamma M^2},
\end{aligned}$$

which are simply the limiting values of the initial conditions. The domain was decomposed into four structured quad meshes with spacings of $h = 1/3$, $h = 2/7$, $h = 1/4$, and $h = 2/9$.

Following Vincent et al. [23] the initial conditions were laid onto the mesh using a collocation projection with $\wp = 3$. The simulation was then run with three different flux reconstruction schemes: DG, SD, and HU as defined in [23]. Solution points were placed at a tensor product construction of Gauss-Legendre quadrature points. Common interface fluxes were computed using a Rusanov Riemann solver. To advance the solutions in time a classical fourth order Runge–Kutta method (RK4) was used. The time step was taken to be $\Delta t = 0.00125$ with $t = 0..1800$. Solutions were written out to disk every 32 000 steps. The order of accuracy of the scheme at a particular time can be determined by plotting $\log \sigma$ against $\log h$ and performing a least squares fit through

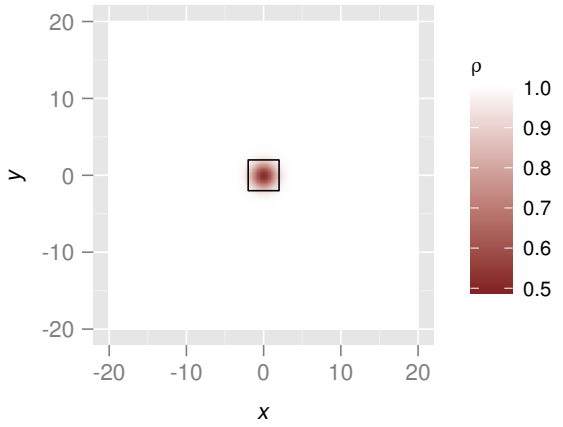


Figure 6.1. Initial density profile for the vortex in Ω.

the four data points. The order is given by the gradient of the fit. A plot of order of accuracy against time for the three schemes can be seen in Figure 6.2. Here the order of accuracy is observed to change as a function of time. This is due to the fact that the error is actually of the form $\sigma(t) = \sigma_{\mathrm{p}} + \sigma_{\mathrm{so}}(t)$ where σ_{p} is a constant projection error and σ_{so} is the time-dependent spatial operator error of (4.14). The projection error arises as a consequence of the collocation projection of the initial conditions onto the mesh. Over time the spatial operator error grows in magnitude and eventually dominates. Only when $\sigma_{\mathrm{so}}(t) \gg \sigma_{\mathrm{p}}$ can the true order of the method be observed. The results here can be seen to be in excellent agreement with those of [23].

6.2 Couette Flow

Consider the case in which two parallel plates of infinite extent are separated by a distance H in the y direction. Both plates are treated as isothermal walls at a temperature T_w with the top plate being permitted to move at a velocity v_w in the x direction with respect to the bottom plate. For simplicity the ordinate of the bottom plate is taken as zero. In the case of a constant viscosity μ the Navier–Stokes equations admit an

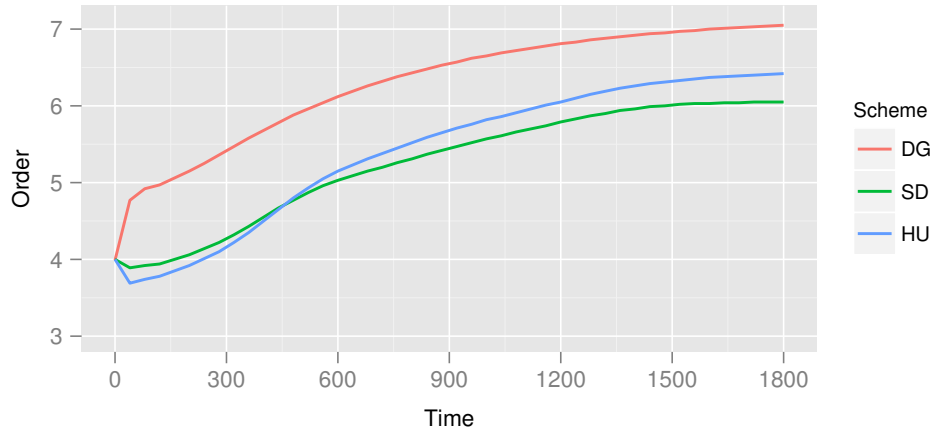


Figure 6.2. Spatial super accuracy observed for a $\wp = 3$ simulation using DG, SD and HU schemes. Results obtained using PyFR v0.1.0.

analytical solution in which

$$\rho(\phi) = \frac{\gamma}{\gamma - 1} \frac{2p}{2c_p T_w + P_r v_w^2 \phi(1 - \phi)}, \tag{6.1}$$

$$\mathbf{v}(\phi) = v_w \phi \hat{\mathbf{x}}, \tag{6.2}$$

$$p = p_c, \tag{6.3}$$

where $\phi = y/H$ and p_c is a constant pressure. The total energy is given by the ideal gas law of (2.45). On a finite domain the Couette flow problem can be modelled through the imposition of periodic boundary conditions. For a two dimensional mesh periodicity is enforced in x whereas for three dimensional meshes it is enforced in both x and z. For the purposes of this experiment the initial conditions were taken as $\gamma = 1.4$, $P_r = 0.72$, $\mu = 0.417$, $c_p = 1005\,\mathrm{J\,K^{-1}}$, $H = 1\,\mathrm{m}$, $T_w = 300\,\mathrm{K}$, $p_c = 1 \times 10^5\,\mathrm{Pa}$, and $v_w = 69.445\,\mathrm{m\,s^{-1}}$. These values correspond to a Mach number of 0.2 and a Reynolds number of 200. The plates were modelled as no-slip isothermal walls as detailed in §B.3 of Appendix B. A plot of the resulting energy profile can be seen in Figure 6.3. Constant initial conditions are taken as $\rho = \langle \rho(\phi) \rangle$, $\mathbf{v} = v_w \hat{\mathbf{x}}$, and $p = p_c$. Using the

analytical solution an L^2 error can be defined as

$$\begin{aligned}\sigma(t)^2 &= \int_{\mathbf{\Omega}} \left[E^{\delta}(\mathbf{x},t) - E(\mathbf{x})\right]^2 \mathrm{d}^{N_D}\mathbf{x} \\ &= \int_{\mathbf{\Omega}_{ei}} \left[E^{\delta}_{ei}(\tilde{\mathbf{x}},t) - E(\boldsymbol{\mathcal{M}}_{ei}(\tilde{\mathbf{x}}))\right]^2 J_{ei}(\tilde{\mathbf{x}})\,\mathrm{d}^{N_D}\tilde{\mathbf{x}} \\ &\approx \left[E^{\delta}_{ei}(\tilde{\mathbf{x}}_{ej},t) - E(\boldsymbol{\mathcal{M}}_{ei}(\tilde{\mathbf{x}}_{ej}))\right]^2 J_{ei}(\tilde{\mathbf{x}}_{ej})\omega_{ej},\end{aligned} \tag{6.4}$$

where $\mathbf{\Omega}$ is the computational domain, $E^{\delta}(\mathbf{x},t)$ is the numerical total energy, and $E(\mathbf{x})$ the analytic total energy. In the third step each integral has been approximated by using a quadrature rule with abscissa $\{\tilde{\mathbf{x}}_{ej}\}$ and weights $\{\omega_{ej}\}$ inside of an element type e. Couette flow is a steady state problem and so in the limit of $t \to \infty$ the numerical total energy should converge to a solution. Using PyFR v0.1.0 and starting from a constant initial condition the L^2 error was computed every 0.1 time units. The simulation was said to have converged when $\sigma(t)/\sigma(t+0.1) \leq 1.01$ where σ is the L^2 error. The time at which this occurs is denoted as t_∞.

Once the system has converged for a range of meshes it is possible to compute the order of accuracy of the scheme. For a given $\wp$ this is the slope of a linear least squares fit of $\log h \sim \log \sigma(t_\infty)$ where h is an approximation of the characteristic grid spacing. The expected order of accuracy is $\wp + 1$. In all simulations inviscid fluxes were computed using the Rusanov approach and the LDG parameters were taken to be $\beta = 1/2$ and $\tau = 0.1$. All simulations were performed with DG correction functions and at double precision. Inside tensor product elements Gauss-Legendre solution and flux points were employed. Triangular elements utilised Williams-Shunn solution points and Gauss-Legendre flux points.

Two dimensional unstructured mixed mesh. For the two dimensional test cases the computational domain was taken to be $[-1,1]\times[0,1]$. This domain was then meshed with both triangles and quadrilaterals at four different refinement levels. The Couette flow problem described above was then solved on each of these meshes. Experimental L^2 errors and orders of accuracy can be seen in Table 6.1. In all cases the expected order of accuracy is obtained.

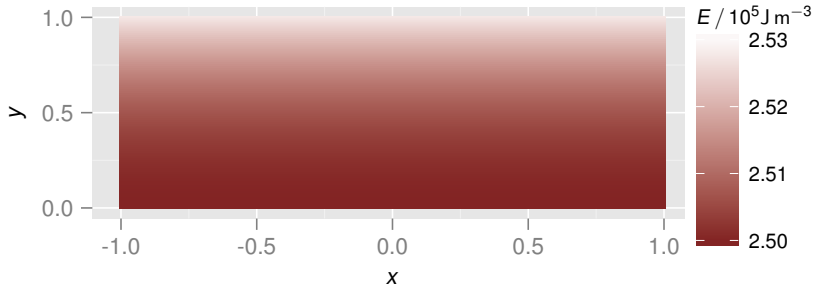


Figure 6.3. Converged steady state energy profile for the two dimensional Couette flow problem.

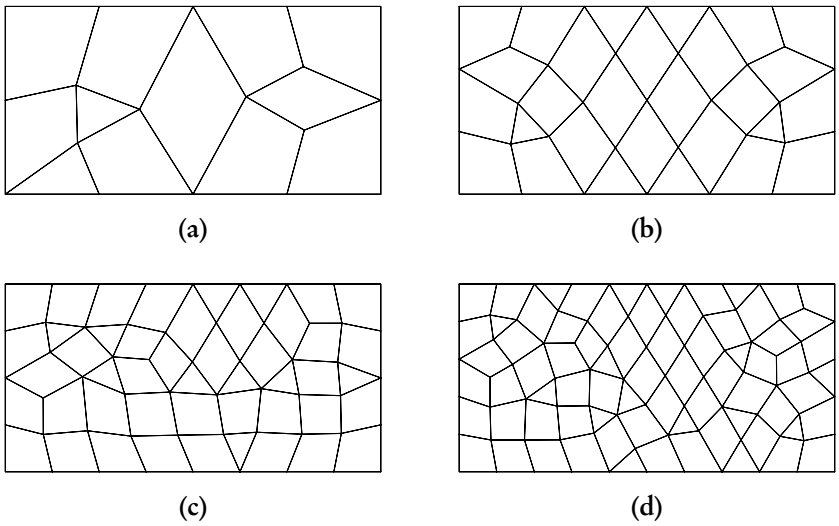


Figure 6.4. Unstructured mixed element meshes used for the two dimensional Couette flow problem.

Table 6.1. L^2 energy error and orders of accuracy for the Couette flow problem on four mixed meshes. The mesh spacing was approximated as $h \sim N_E^{-1/2}$ where N_E is the total number of elements in the mesh.

		$\sigma(t_\infty)$ / J m^{-3}			
Tris	Quads	$\wp = 1$	$\wp = 2$	$\wp = 3$	$\wp = 4$
2	8	1.26×10^{2}	5.77×10^{-1}	5.54×10^{-3}	6.62×10^{-5}
6	22	3.56×10^{1}	1.40×10^{-1}	6.72×10^{-4}	3.91×10^{-6}
10	37	2.08×10^{1}	4.35×10^{-2}	2.54×10^{-4}	8.16×10^{-7}
16	56	1.46×10^{1}	3.52×10^{-2}	1.09×10^{-4}	4.62×10^{-7}
	Order	2.21 ± 0.12	2.99 ± 0.32	3.97 ± 0.05	5.20 ± 0.38

Three dimensional extruded hexahedral mesh. For this three dimensional case the computational domain was taken to be $[-1, 1] \times [0, 1] \times [0, 1]$. Meshes were constructed through first generating a series of unstructured quadrilateral meshes in the *x*-*y* plane. A three layer extrusion was then performed on these meshes to yield a series of hexahedral meshes. Experimental L^2 errors and orders of accuracy for these meshes can be seen in Table 6.2.

Three dimensional unstructured hexahedral mesh. As a further test a domain of dimension $[0, 1]^3$ was considered. This domain was meshed using completely unstructured hexahedra. Three levels of refinement were used resulting in meshes with 96, 536, and 1 004 elements. A cutaway of the most refined mesh can be seen in Figure 6.5. Experimental L^2 errors and the resulting orders of accuracy are presented in Table 6.3. Despite the fully unstructured nature of the mesh the expected order of accuracy was again obtained in all cases. It is, however, worth noting the higher standard errors associated with these results.

Table 6.2. L^2 energy errors and orders of accuracy for the Couette flow problem on three extruded hexahedral meshes. On account of the extrusion $h \sim N_E^{-1/2}$ where N_E is the total number of elements in the mesh.

	$\sigma(t_\infty)$ / J m^{-3}		
Hexes	$\wp = 1$	$\wp = 2$	$\wp = 3$
78	3.35×10^{1}	5.91×10^{-2}	7.28×10^{-4}
195	1.23×10^{1}	1.87×10^{-2}	1.15×10^{-4}
405	6.15×10^{0}	5.49×10^{-3}	2.72×10^{-5}
Order	2.06 ± 0.08	2.87 ± 0.24	3.99 ± 0.03

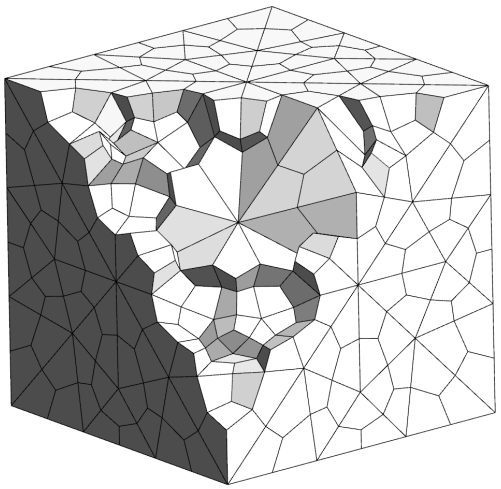


Figure 6.5. Cutaway of the unstructured hexahedral mesh with 1 004 elements.

Table 6.3. L^2 energy errors and orders of accuracy for the Couette flow problem on three unstructured hexahedral meshes. Mesh spacing was taken as $h \sim N_E^{-1/3}$ where N_E is the total number of elements in the mesh.

	$\sigma(t_\infty)$ / J m^{-3}		
Hexes	$\wp = 1$	$\wp = 2$	$\wp = 3$
96	1.91×10^{1}	4.32×10^{-2}	5.83×10^{-4}
536	8.20×10^{0}	9.11×10^{-3}	6.89×10^{-5}
1004	3.82×10^{0}	3.22×10^{-3}	2.04×10^{-5}
Order	1.93 ± 0.46	3.19 ± 0.48	4.16 ± 0.44

6.3 Cylinder Flow at $Re = 3\,900$

Flow over a circular cylinder has been the focus of various experimental and numerical studies [74–80]. Characteristics of the flow are known to be highly dependent on the Reynolds number Re, defined as

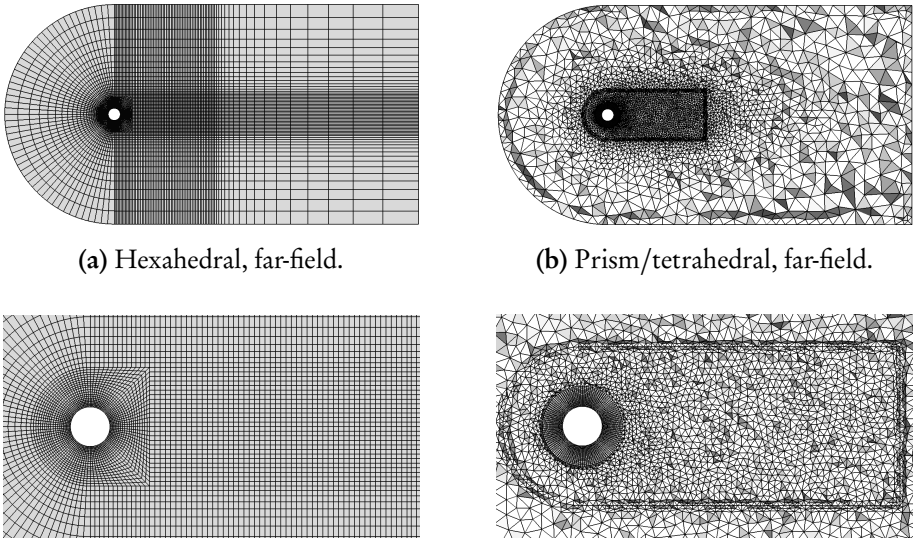
$$Re = \frac{u_\infty D}{\nu}, \tag{6.5}$$

where u_∞ is the free-stream fluid speed, D is the cylinder diameter, and ν is the fluid kinematic viscosity. Roshko [81] identified a stable range between $Re = 40$ and 150 that is characterised by the shedding of regular laminar vortices, as well as a transitional range between $Re = 150$ and 300, and a turbulent range beyond $Re = 300$. These results were subsequently confirmed by Bloor [82], who identified a similar set of regimes. Later, Williamson [83] identified two modes of transition from two dimensional to three dimensional flow. The first, known as Mode-A instability, occurs at $Re \approx 190$ and the second, known as Mode-B instability, occurs at $Re \approx 260$. The turbulent range beyond $Re = 300$ can be further sub-classified into the shear-layer transition, critical, and supercritical regimes as discussed in the review by Williamson [84].

In the present study [85] flow over a circular cylinder at $Re = 3\,900$ with an effectively incompressible Mach number of 0.2 is considered. This case sits in the shear-layer transition regime identified by Williamson [84], and contains several complex flow features, including separated shear layers, turbulent transition, and a fully turbulent wake. This test case has been the focus of a number of previous studies, both experimental and numerical [76–80]. Recently, Lehmkuhl et al. [86] demonstrated that the wake profile for this test case can be classified as one of two modes, a low-energy mode (Mode-L) and a high-energy mode (Mode-H). Specifically, via analysis of a very long period simulation of over 2 000 convective times, they showed that the wake fluctuates between these two modes.

Domain. In the present study a computational domain with dimensions $[-9D, 25D]$; $[-9D, 9D]$; and $[0, \pi D]$ in the stream-, cross-, and span-wise directions, respectively, is used. The cylinder is centred at $(0, 0, 0)$. The span-wise extent was chosen based on the results of Norberg [76], who found no significant influence on statistical data when the span-wise dimension was doubled from πD to $2\pi D$. Indeed, a span of πD has been used in the majority of previous numerical studies [77–79], including the recent DNS study of Lehmkuhl et al. [86]. The stream-wise and cross-wise dimensions are comparable to the experimental and numerical values used by Parnaudeau et al. [80], whose results will be directly compared with those computed by PyFR. The overall domain dimensions are also comparable to those used for DNS studies by Lehmkuhl et al. [86]. The domain is periodic in the span-wise direction, with the no-slip isothermal wall boundary condition of §B.3 applied at the surface of the cylinder and a Riemann invariant boundary condition, as detailed in §B.4, applied at the far-field.

Meshes. The domain was meshed in two ways. The first mesh consisted of entirely structured hexahedral elements, whilst the second was unstructured, consisting of prismatic elements in the near wall boundary layer region, and tetrahedral elements in the wake and far-field. Both meshes employed quadratically curved elements, and were designed to fully resolve the near wall boundary layer region when $\wp = 4$. Specifically, the maximum skin friction coefficient was estimated *a priori* as $C_f \approx 0.075$ based



(a) Hexahedral, far-field.

(b) Prism/tetrahedral, far-field.

(c) Hexahedral, wake.

(d) Prism/tetrahedral, wake.

Figure 6.6. Cutaways through the two meshes.

on the LES results of Breuer [78]. The height of the first element was then specified such that when $\wp = 4$ the first solution point from the wall sits at $y+ \approx 1$, where non-dimensional wall units are calculated in the usual fashion as $y+ = u_\tau y/\nu$ with $u_\tau = \sqrt{C_f/2}u_\infty$.

The hexahedral mesh had 104 elements in the circumferential direction, and 16 elements in the span-wise direction, which when $\wp = 4$ achieves span-wise resolution comparable to that used in previous studies, as discussed by Breuer [78] and the references contained therein. The prism/tetrahedral mesh has 116 elements in the circumferential direction, and 20 elements in the span-wise direction, these numbers being chosen to help reduce face aspect ratios at the edges of the prismatic layer; which facilitates transition to the fully unstructured tetrahedral elements in the far-field. In total the hexahedral mesh contained 119 776 elements, and the prism/tetrahedral mesh contained 79 344 prismatic elements and 227 298 tetrahedral elements. Both meshes are shown in Figure 6.6.

Table 6.4. Approximate memory requirements of PyFR for the two cylinder meshes.

	Device memory / GiB			
Mesh	$\wp = 1$	$\wp = 2$	$\wp = 3$	$\wp = 4$
Hex	0.8	2.1	4.1	7.3
Pri/tet	1.1	2.6	4.7	7.7

Methodology. The compressible Navier–Stokes equations, with constant viscosity, were solved on each of the two meshes shown in Figure 6.6 using PyFR v0.2.4. A DG scheme was used for the spatial discretisation, a Rusanov Riemann solver was used to calculate the inviscid fluxes at element interfaces, and the explicit RK45[2R+] scheme of Carpenter and Kennedy [20] was used to advance the solution in time. No sub-grid model was employed, hence the approach should be considered ILES/DNS [87, 88], as opposed to classical LES. The approximate memory requirements of PyFR for these simulations with different $\wp$ are detailed in Table 6.4. The total required floating point operations per RK45[2R+] time-step with different $\wp$ are detailed in Figure 6.7. When running with $\wp = 1$ both meshes require $\sim 1.5 \times 10^{10}$ floating point operations per time-step. This number can be seen to increase rapidly with $\wp$.

Accuracy. In this section instantaneous and time-span-averaged—henceforth referred to as averaged—results obtained using a cluster of 12 NVIDIA K20c GPUs at $\wp = 4$ are presented. Both simulations are run for 1 000 convective times, allowing the flow to fluctuate between Mode-H and Mode-L as identified by Lehmkuhl et al. [77, 86]. A moving window time-average with a width of 100 convective times is used to extract the modes from the long-period simulation. This yields four datasets including both Mode-H and Mode-L for both the hexahedral and prism/tetrahedral meshes. Both modes are then compared with results from previous experimental and numerical studies, where either one or both of the modes were observed [76, 77, 80, 86].

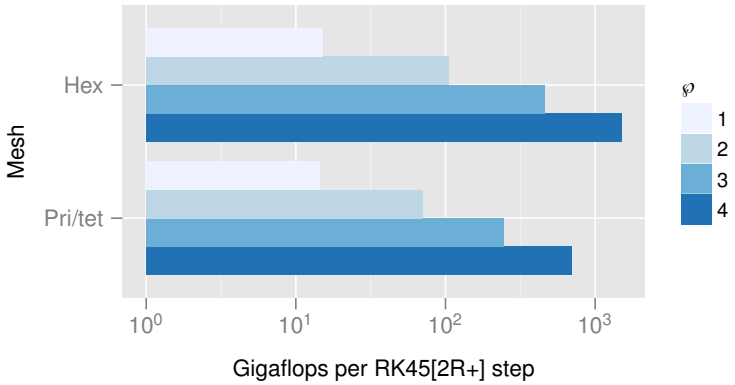
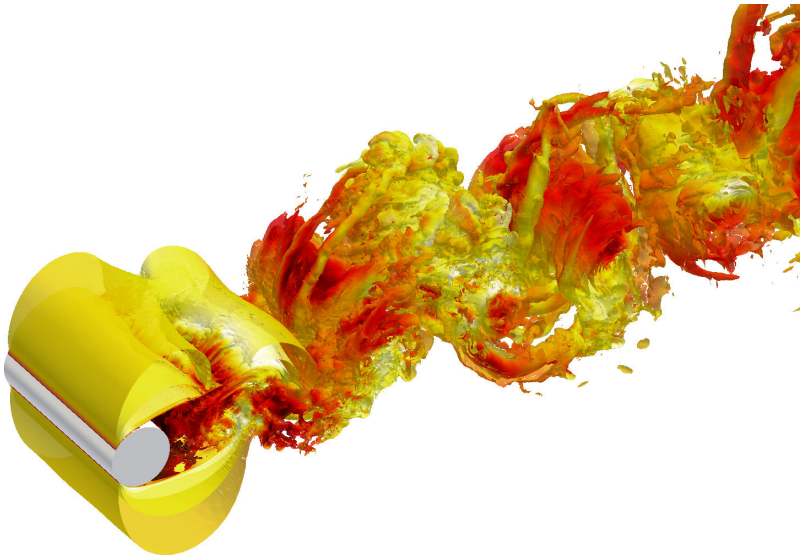


Figure 6.7. Computational effort required for the 119 776 element hexahedral mesh and the mixed mesh with 79 344 prims and 227 298 tetrahedra.

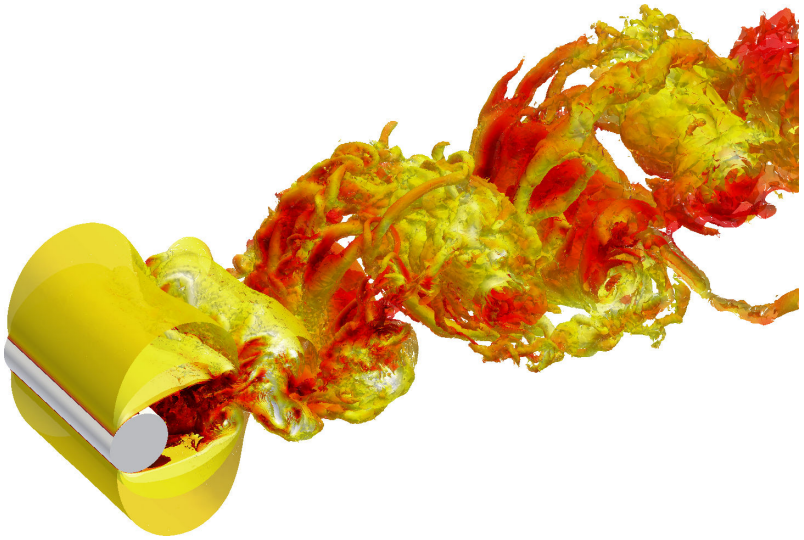
Instantaneous surfaces of iso-density are shown in Figure 6.8 for both simulations at similar phases of the shedding cycle. Laminar flow is observed at the leading edge of the cylinder for both test cases with a turbulent transition occurring near the separation points, and finally fully turbulent flow is found in the wake region. These are the characteristic features of the shear-layer transition regime, as described by Williamson [84]. The wake is composed of large vortices, alternately shedding off from the upper and lower surfaces of the cylinder, and smaller scale turbulent structures.

Plots of the averaged stream-wise wake profiles are shown in Figure 6.9 and Figure 6.10 for Mode-H and Mode-L, respectively. Both meshes show excellent agreement with the numerical results of Lehmkuhl et al. [86] for both modes and with the experimental results of Parnaudeau et al. [80], which is available for Mode-L. The Mode-H cases exhibit relatively shorter separation bubbles and the Mode-L cases have characteristic inflection points in the wake profile near $x/D \approx 1$.

Plots of the averaged pressure coefficient $\overline{C_p}$ on the surface of the cylinder are shown in Figure 6.11 and Figure 6.12 for both extracted modes and both meshes. The Mode-H results are shown alongside the Mode-H numerical results of Lehmkuhl et al. [86] and the results from Case I of Ma et al. [77]. The Mode-L results are shown alongside the Mode-L numerical results of Lehmkuhl et al. [86] and the experimental



(a) Structured hexahedral.



(b) Unstructured mixed prism/tetrahedral.

Figure 6.8. Instantaneous surfaces of iso-density coloured by velocity magnitude.

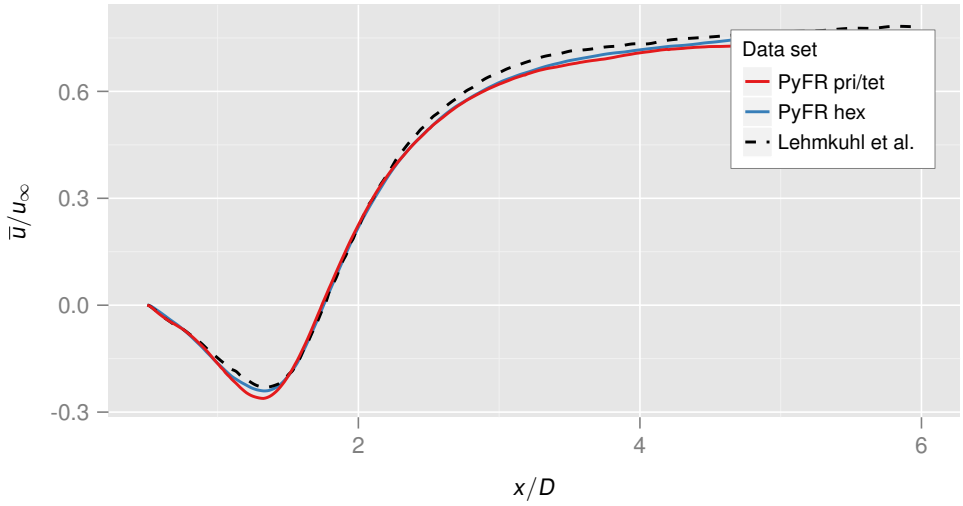


Figure 6.9. Averaged wake profiles for Mode-H compared with the numerical results of Lehmkuhl et al. [86].

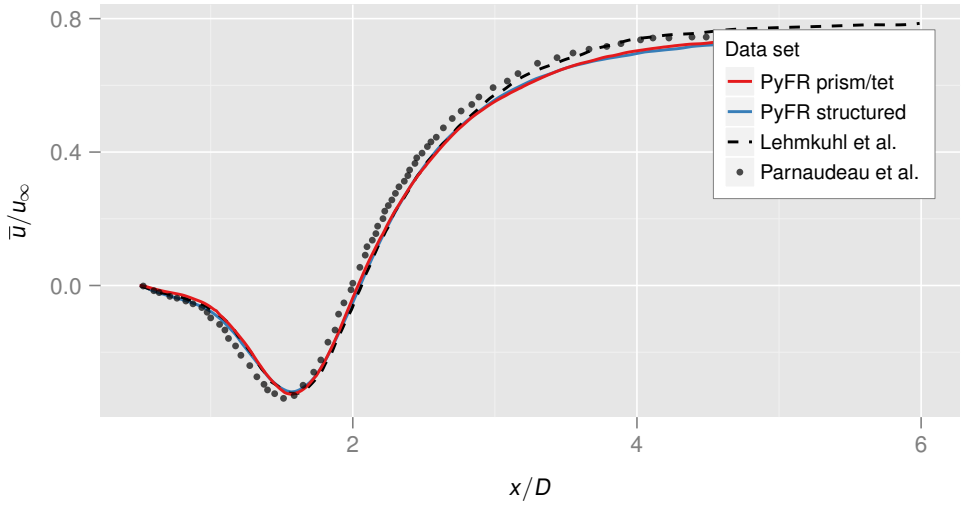


Figure 6.10. Averaged wake profiles for Mode-L compared with the numerical results of Lehmkuhl et al. [86] and experimental results of Parnaudeau et al. [80].

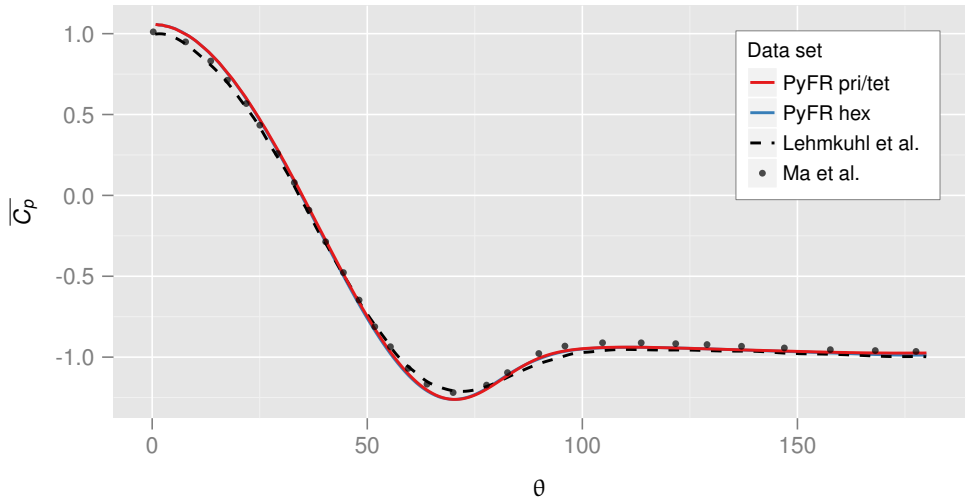


Figure 6.11. Averaged pressure coefficient for Mode-H compared with the numerical results of Ma et al. [77] and Lehmkuhl et al. [86].

results of Norberg et al. at a similar $Re = 4\,020$ [76], which were extracted from Kravchenko and Moin [79]. Both modes have similar pressure coefficient distributions at the leading face of the cylinder, while the Mode-H case has stronger suction on the trailing face adjacent to the separation bubble. Both modes extracted using both meshes show excellent agreement with their corresponding reference data sets.

The averaged pressure coefficient at the base of the cylinder $\overline{C_{pb}}$, and the averaged separation angle θ_s measured from the leading stagnation point are tabulated in Table 6.5 for both modes and meshes. These are shown along with measurements from the experimental results of Norberg et al. [76], experimental data from Parnaudeau et al. [80], and DNS data from Lehmkuhl et al. [86] for both modes. Both measured quantities agree well with the reference data sets for both modes and meshes. The difference in separation angle is less than $\sim 1^\circ$ between the current and reference results. The pressure coefficient at the base of the cylinder shows that the high-energy Mode-H case has stronger recirculation in the wake, characterised by greater suction at the wall adjacent to the recirculation bubble.

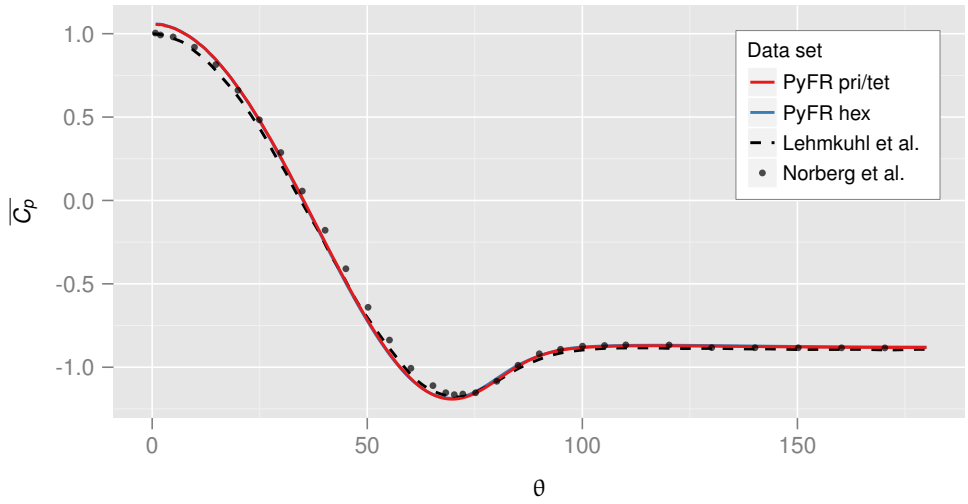


Figure 6.12. Averaged pressure coefficient for Mode-L compared with the numerical results of Lehmkuhl et al. [86] and experimental results of Norberg et al. [76].

Table 6.5. Comparison of quantitative values with experimental and DNS results.

	Mode-H		Mode-L	
	$-\overline{C_{pb}}$	$\theta_s/°$	$-\overline{C_{pb}}$	$\theta_s/°$
PyFR hex	0.987	88.28	0.880	87.71
PyFR pri/tet	0.974	87.13	0.882	86.90
Parnaudeau et al. [80]				88.00
Lehmkuhl et al. [86]	0.980	88.25	0.877	87.80
Norberg et al. [76, 79]			0.880	

Plots of averaged stream-wise velocity at $x/D = 1.06$, 1.54, and 2.02 are shown in Figure 6.13 and Figure 6.14 for the Mode-H and Mode-L simulations, respectively. These results are shown alongside the experimental results of Parnaudeau et al. [80] for Mode-L, the numerical results of Ma et al. [77] for Mode-H, and the DNS results of Lehmkuhl et al. [86] for both modes. Both the simulations show the V-shaped velocity profile for Mode-H at $x/D = 1.06$ and the U-shaped profile for Mode-L, also at $x/D = 1.06$. Both modes on both meshes agree well with both their corresponding reference data sets. Plots of averaged cross-wise velocity at $x/D = 1.06$, 1.54, and 2.02 are shown in Figure 6.15 and Figure 6.16, respectively. These cross-wise velocity profiles also show excellent agreement with their corresponding reference data sets.

6.4 Single-Node Performance

In this section the performance of PyFR on an Intel Xeon E5-2697 v2 CPU, an NVIDIA Tesla K40c GPU, and an AMD FirePro W9100 GPU is analysed. Various attributes of the E5-2697, K40c, and W9100 are detailed in Table 6.6. The theoretical peaks for double precision arithmetic and memory bandwidth were obtained from vendor specifications. However, in practice it is usually only possible to obtain these theoretical peak values using specially crafted code sequences. Such sequences are almost always platform specific and seldom perform useful computations. Consequently, the *reference* peaks are also calculated and tabulated. Reference peaks for double precision arithmetic are defined here as the maximum number of giga-floating point operations per second (GFLOP/s) obtained while multiplying two large double precision row-major matrices using DGEMM from an appropriate BLAS library. Reference peaks for memory bandwidth are defined here as the rate, in gigabytes per second (GB/s), that data can be copied between two one gigabyte buffers. Reference peaks for the E5-2697 were obtained using DGEMM from the Intel Math Kernel Libraries (MKL) version 11.1.2, and with the E5-2697 paired with four DDR3-1600 DIMMs (with Turbo Boost enabled). Reference peaks for the K40c were obtained using DGEMM from cuBLAS as shipped with CUDA 6, with GPU boost disabled and ECC enabled. Reference peaks

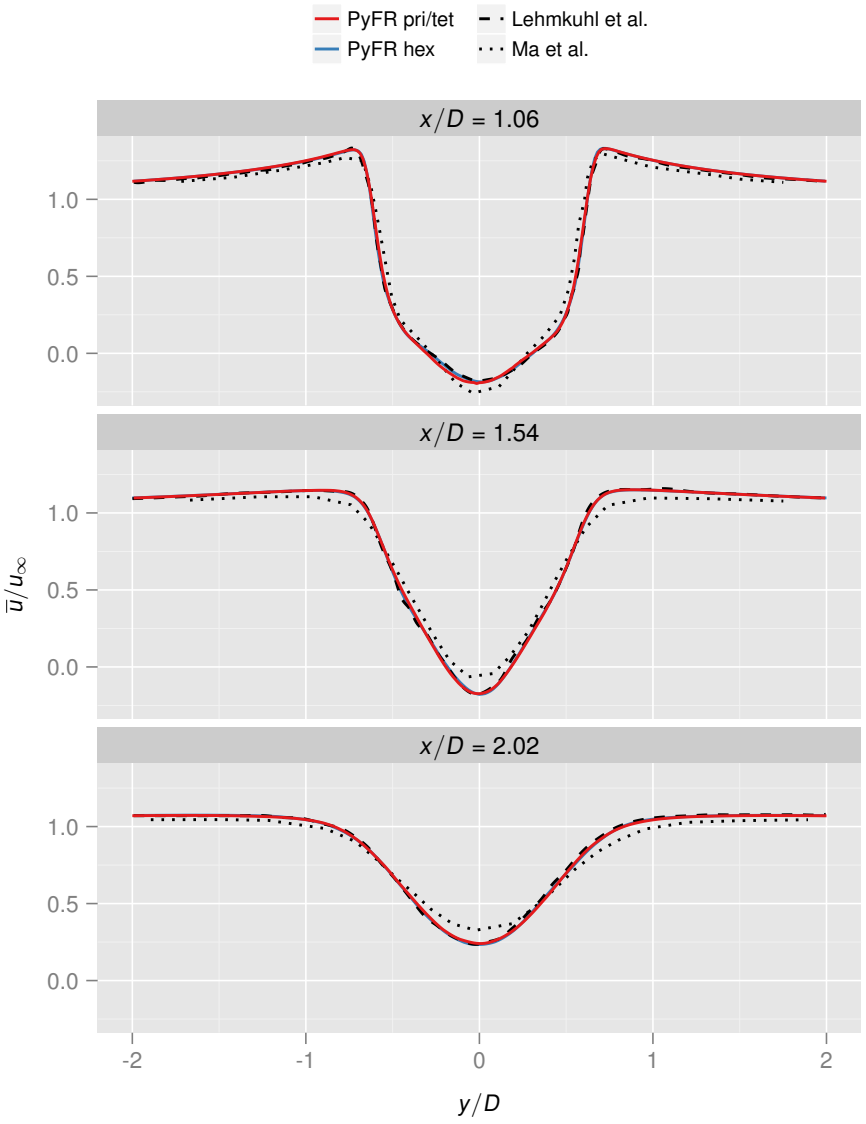


Figure 6.13. Time-span-average stream-wise velocity profiles for Mode-H compared with the numerical results of Lehmkuhl et al. [86] and Ma et al. [77].

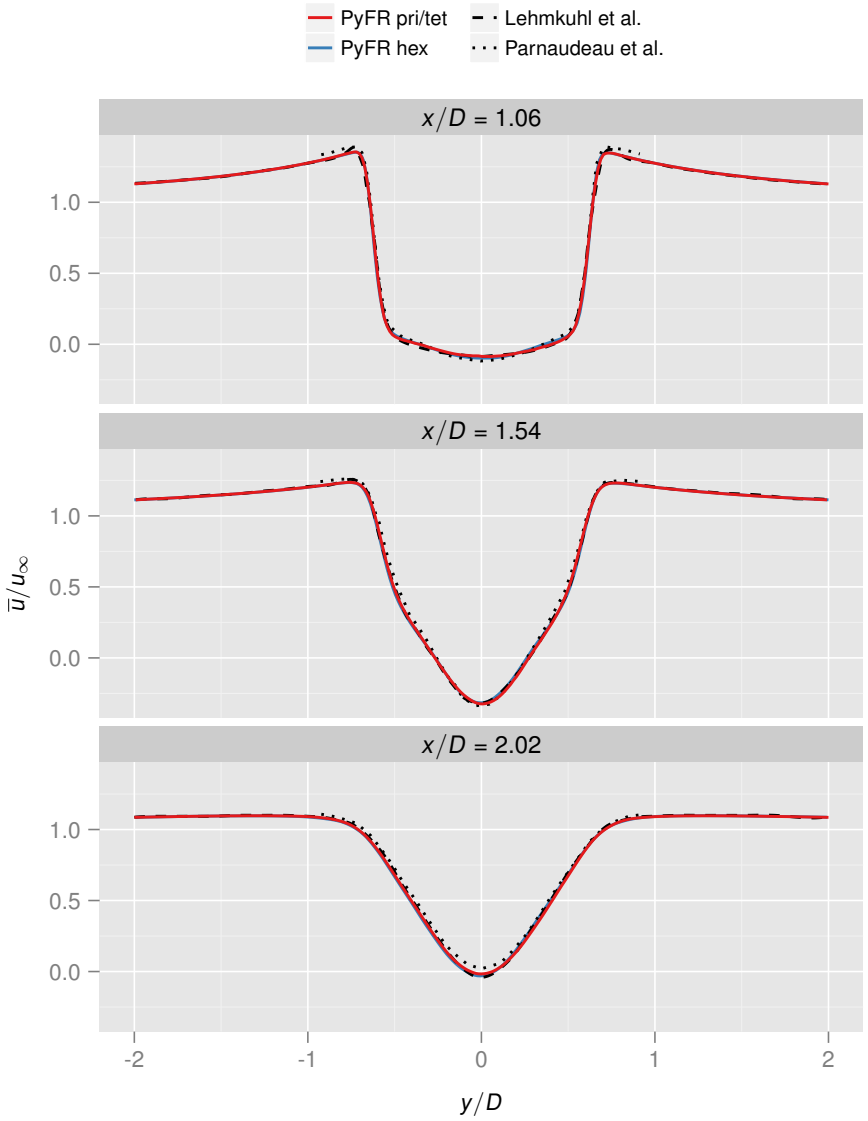


Figure 6.14. Time-span-average stream-wise velocity profiles for Mode-L compared with the numerical results of Lehmkuhl et al. [86] and experimental results of Parnaudeau et al. [80].

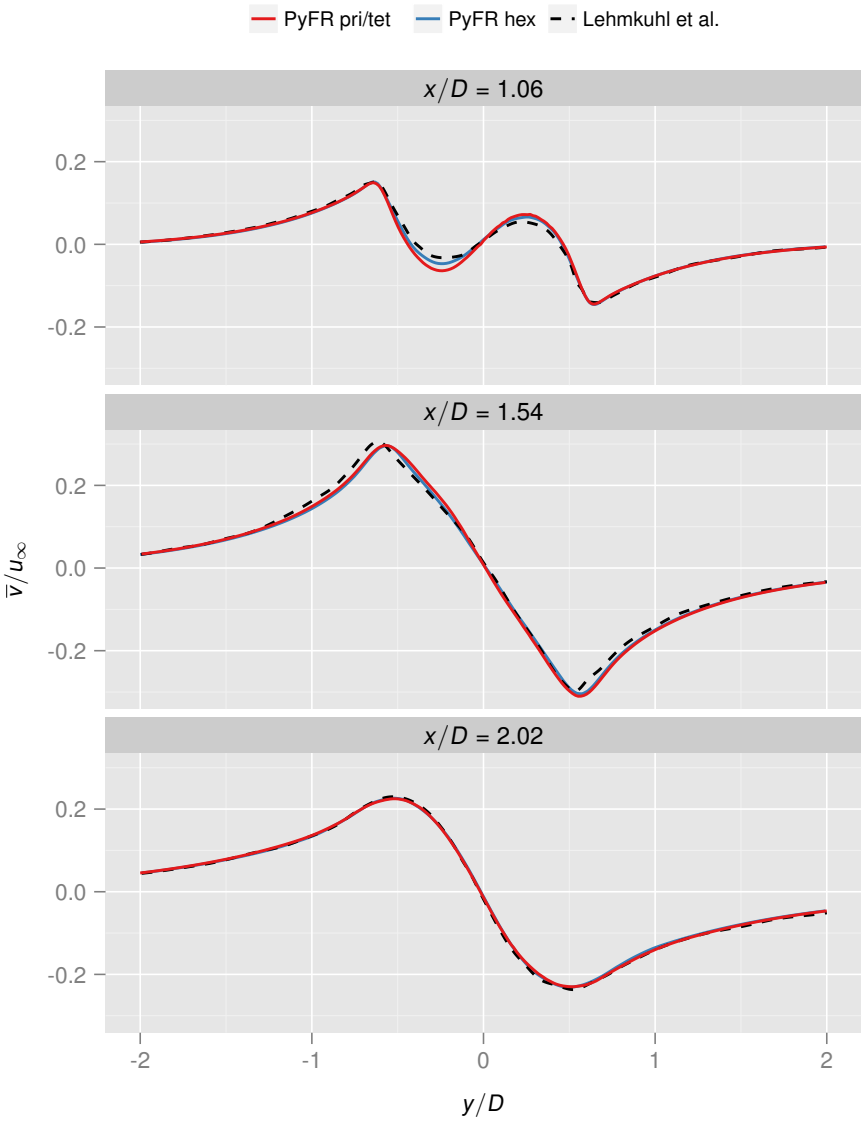


Figure 6.15. Time-span-average cross-stream velocity profiles for Mode-H compared with the numerical results of Lehmkuhl et al. [86].

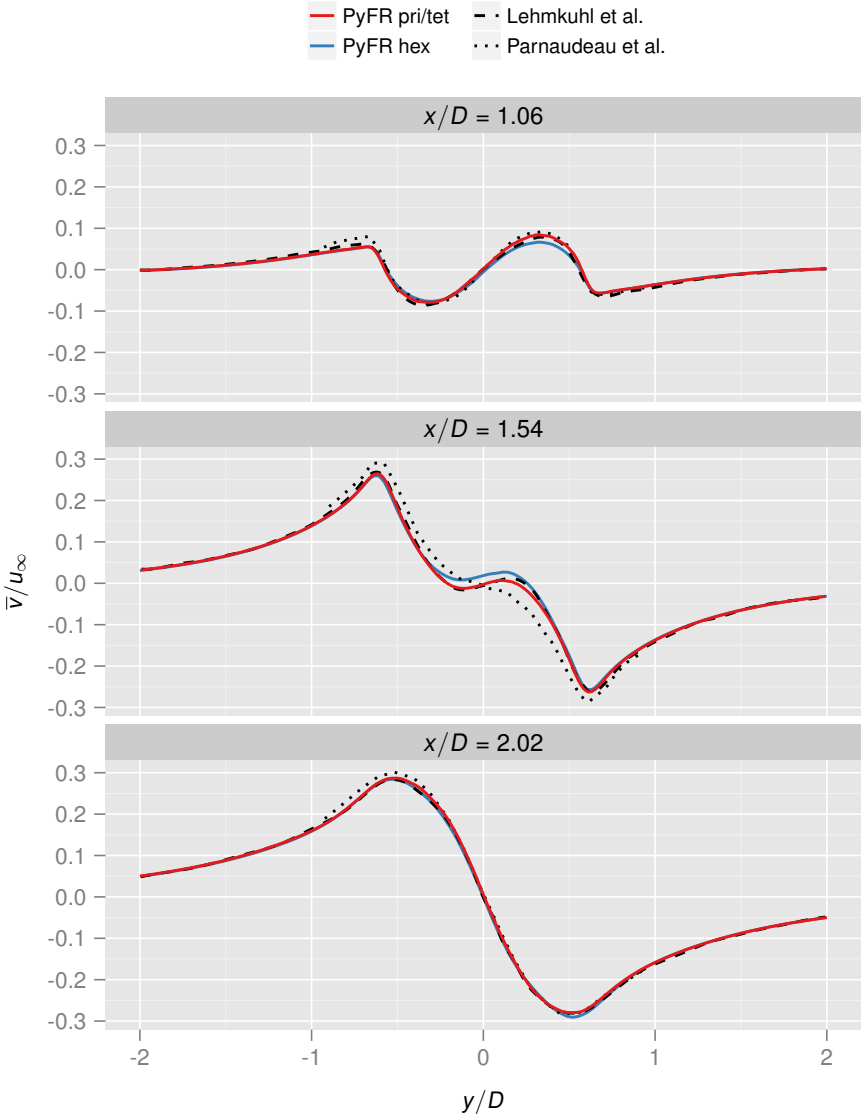


Figure 6.16. Time-span-average cross-stream velocity profiles for Mode-L compared with the numerical results of Lehmkuhl et al. [86] and experimental results of Parnaudeau et al. [80].

for the W9100 were obtained using DGEMM from clBLAS v2.0 with version 1411.4 of the AMD APP OpenCL runtime.

On the K40c ECC is implemented in software and hence when enabled error-correction data is stored in global memory. A consequence of this is that when ECC is enabled there is a reduction in available memory and memory bandwidth. This partially accounts for the discrepancy observed between the theoretical and reference memory bandwidths for the K40c. For both the K40c and the E5-2697, reference peaks for double precision arithmetic are in excess of 80% of their theoretical values. However, for the W9100 the reference peak for double precision arithmetic is only 34% of its theoretical value. This value is not significantly improved via the auto-tuning utility that ships with clBLAS. It is hoped that this figure will improve with future releases of clBLAS.

In preparing Table 6.6 the decision has been made to deliberately omit the number of 'cores' available on each platform. This is on account of the term being both ill-defined and routinely subject to abuse in the literature. For example, the E5-2697 is presented by Intel as having 12 cores, whereas the K40c is described by NVIDIA as having 2880 'CUDA cores'. However, whereas the cores in the E5-2697 can be considered linearly independent those in the K40c can not. The rough equivalent of a CPU core in NVIDIA parlance is a 'streaming multiprocessor', or SMX, of which the K40c has 15. Additionally, the E5-2697 has support for two-way simultaneous multithreading—referred to by Intel as Hyper-Threading—permitting two threads to execute on each core. At any one instant it is therefore possible to have up to 24 independent threads resident on a single E5-2697. The AMD equivalent of a CUDA core is a 'stream processor' of which the W9100 has 2816. This is not to be confused with the aforementioned streaming multiprocessor of NVIDIA; for which the AMD equivalent is a 'Compute Unit'. Practically, both CUDA cores and stream processors are closer to the individual vector lanes of a traditional CPU core. Given this minefield of confusing nomenclature the choice has instead been made to just state the peak floating point capabilities of the hardware.

Table 6.6. Baseline attributes of the three hardware platforms. For the NVIDIA Tesla K40c GPU Boost was left disabled and ECC was enabled. The Intel Xeon E5-2697 v2 was paired with four DDR3-1600 DIMMs with Turbo Boost enabled.

	Platform		
	K40c	W9100	E5-2697
Arithmetic / GFLOP/s			
theoretical peak	1430	2620	280
reference peak	1192	890	231
Memory bandwidth / GB/s			
theoretical peak	288	320	51.2
reference peak	190	261	37.1
Thermal design power / W	235	275	130
Memory / GiB	12	16	
Clock / MHz	745	930	3000
Transistors / Billion	7.1	6.2	4.3

Results and discussion. By measuring the wall clock time required for PyFR to take 500 RK45[2R+] time-steps, and utilising the operation counts per time-step detailed in Figure 6.7, one can calculate the sustained performance of PyFR in GFLOP/s when running with the meshes detailed in Figure 6.6 with $\wp = 1, 2, 3, 4$.

Sustained performance of PyFR for the various hardware platforms is shown in Figure 6.17. From the figure it is clear that the computational efficiency of PyFR increases with the polynomial order. This is consistent with higher order simulations having an increased compute intensity per degree of freedom. This additional intensity results in larger operator matrices that are better suited to the tiling schemes employed by BLAS libraries. The OpenCL implementation shipped by NVIDIA as part of CUDA only supports the use of 32-bit memory pointers. As such a single context is limited to

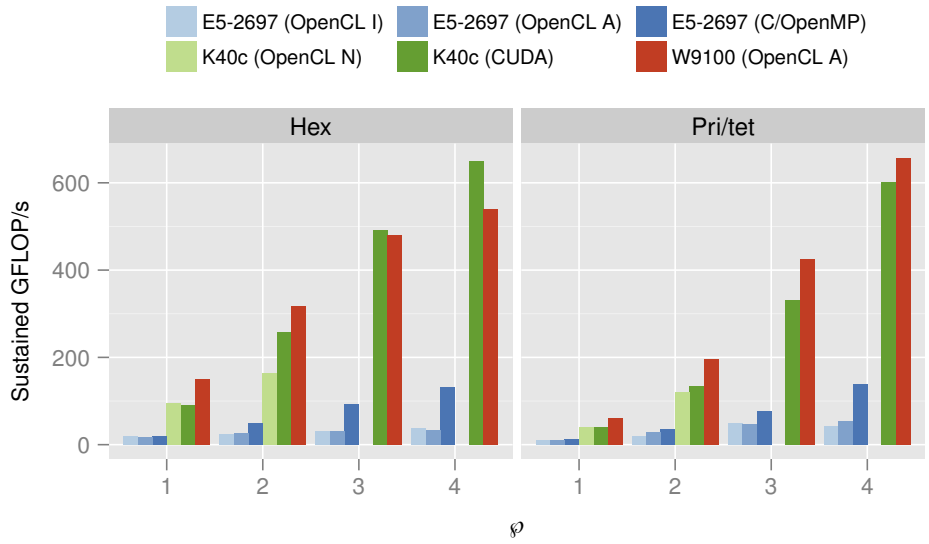


Figure 6.17. Sustained performance of PyFR in GFLOP/s for the various pieces of hardware. The backend used by PyFR is given in parentheses. For the OpenCL backend the initial of the vendor is suffixed. As the NVIDIA OpenCL platform is limited to 4 GiB of memory no results are available for $\wp = 3, 4$.

4 GiB of memory, *cf.* Table 6.4. It was therefore not possible to perform the third and fourth order simulations for either of the two meshes using the OpenCL backend with the K40c.

The Intel and AMD implementations of OpenCL, when used in conjunction with clBLAS, are only competitive with the C/OpenMP backend when $\wp = 1$ for the hexahedral mesh, and $\wp = 1, 2$ for the prism/tetrahedral mesh. This is also the case when comparing performance between the CUDA backend and the NVIDIA OpenCL backend on the K40c. Prior analysis by Witherden et al. [89] suggests that at these orders a reasonable proportion of the wall clock time will be spent in the bandwidth-bound pointwise kernels as opposed to DGEMM. On account of being bandwidth-bound such kernels do not extensively test the optimisation capabilities of the compiler. When $\wp = 4$ both implementations of OpenCL on the E5-2697 are delivering between one

third and one quarter of the performance of the native backend. This highlights the lack of performance portability associated with OpenCL in this context, confirming the initial contention that, at the time of writing, performance portability can only be achieved effectively via native paradigms. Further, it also justifies the approach to multi-platform computing that has been adopted within PyFR.

Performance of the K40c culminates at 649 GFLOP/s for the $\wp = 4$ hexahedral mesh. This represents some 45% of the theoretical peak and 54% of the reference peak. By comparison the E5-2697 obtains 132 GFLOP/s for the same simulation equating to 47% and 57% of the theoretical and reference peaks, respectively. Performance does improve slightly to 140 GFLOP/s for the $\wp = 4$ prism/tetrahedral mesh, however. On this same mesh at $\wp = 4$ the W9100 can be seen to sustain 657 GFLOP/s of throughput. Although, in absolute terms, this observation represents the highest sustained rate of throughput it corresponds to just 25% of the theoretical peak. However, working in terms of realisable peaks, PyFR is found to obtain some 74% of the reference value.

The wall clock time required per degree of freedom (DOF) to evaluate $\boldsymbol{\nabla} \cdot \mathbf{f}$ for each simulation can be seen in Table 6.7. The DOF count is inclusive of the factor of five arising from there being five distinct field variables at each solution point. This quantity can be used to evaluate the efficiency of PyFR relative to other codes. With the exception of OpenCL on the E5-2697 the time per DOF reaches a minima for the hexahedral mesh at $\wp = 3$. This shows that as $\wp$ is raised from one to three the increasing number of floating point operations required to update each DOF is being offset by the improving efficiency of PyFR. The pattern is similar for the prism/tetrahedral mesh except that for the E5-2697 (C/OpenMP) and the K40c (CUDA) the minima is at $\wp = 4$.

6.5 Multi-Node Heterogeneous Performance

Having determined the performance characteristics of PyFR on various individual platforms, it is now possible to investigate the ability of PyFR to undertake simulations on a multi-node heterogeneous system containing an Intel Xeon E5-2697 v2 CPU, an NVIDIA Tesla K40c GPU, and an AMD FirePro W9100 GPU. The experimental set

Table 6.7. Time to evaluate $\nabla \cdot f$ normalised by the total number of DOFs.

		Time per DOF / 10^{-9} s			
Mesh	Platform	$\wp = 1$	$\wp = 2$	$\wp = 3$	$\wp = 4$
Hex	E5-2697 (OpenCL I)	32.31	53.04	75.96	106.61
	E5-2697 (OpenCL A)	35.48	49.75	79.40	119.76
	E5-2697 (C/OpenMP)	32.14	28.87	27.74	31.95
	K40c (OpenCL N)	6.51	7.92		
	K40c (CUDA)	6.93	5.05	4.88	6.17
	W9100 (OpenCL A)	4.17	4.08	5.00	7.43
Pri/tet	E5-2697 (OpenCL I)	46.09	60.28	53.07	104.03
	E5-2697 (OpenCL A)	40.37	41.41	53.88	78.17
	E5-2697 (C/OpenMP)	46.32	40.68	36.74	35.53
	K40c (OpenCL N)	12.82	11.15		
	K40c (CUDA)	12.94	10.40	8.61	8.18
	W9100 (OpenCL A)	8.72	7.35	7.11	7.52

up and methodology is the same as the single-node case.

Mesh partitioning. In order to distribute a simulation across the nodes of the heterogeneous system it is first necessary to partition the mesh. High quality partitions can be readily obtained using a graph partitioning package such as METIS [90] or SCOTCH [91].

When partitioning a mixed element mesh for a *homogeneous* cluster it is necessary to suitably weight each element type according to its computational cost. This cost depends both upon the platform on which PyFR is running and the order at which the simulation is being performed. In principle it is possible to measure this cost; however in practice the following set of weights have been found to give satisfactory results across most polynomial orders and platforms

$$\text{hex} : \text{pri} : \text{tet} = 3 : 2 : 1,$$

where larger numbers indicate a greater computational cost. One subtlety that arises here, is that from a graph partitioning standpoint there is no penalty associated with placing a sole vertex (element) of a given weight inside of a partition. Computationally, however, there is a very real penalty incurred from having just a single element of a certain type inside of the partition. It is therefore desirable to avoid mesh partitions where any one partition contains less than around a thousand elements of a given type. An exception is when a partition contains no elements of such a type—in which case zero overheads are incurred.

When partitioning a mesh with one type of element for a *heterogeneous* cluster it is necessary to weight the partition sizes in line with the performance characteristics of the hardware on each node. However, in the case of a mixed element mesh on a heterogeneous cluster the weight of an element is no longer static but rather depends on the partition that it is placed in—a significantly richer problem. Solving such a problem is currently beyond the capabilities of most graph partitioning packages. Accordingly, mixed element meshes that are partitioned for heterogeneous clusters often exhibit inferior load balancing than those partitioned for homogeneous systems. Moreover, for consistent performance it is necessary to dedicate a CPU core to each accelerator in

Table 6.8. Partition weights for the multi-node heterogeneous simulation.

	E5-2697 : W9100 : K40c			
Mesh	$\wp = 1$	$\wp = 2$	$\wp = 3$	$\wp = 4$
Hex	3:27:23	3:27:24	4:24:26	4:24:28
Pri/tet	5:33:17	5:33:17	5:30:20	5:27:23

the system. The amount of useful computation that can be performed by the host CPU is therefore reduced in accordance with this.

Given the single-node performance numbers of Figure 6.17 it comports to pair the E5-2697 with the C/OpenMP backend, the K40c with the CUDA backend, and the W9100 with the OpenCL backend, in order to achieve optimal performance. Employing these results, in conjunction with some light experimentation, a set of partitioning weights were obtained and are tabulated in Table 6.8.

Results and discussion. Sustained performance of PyFR on the multi-node heterogeneous system for each of the meshes detailed in Figure 6.6 with $\wp = 1, 2, 3, 4$ is shown in Figure 6.18. Under the assumptions of perfect partitioning and scaling one would expect the sustained performance of the heterogeneous simulation to be equivalent to the sum of the E5-2697 (C/OpenMP), K40c (CUDA), and W9100 (OpenCL A) bars in Figure 6.17. However, for reasons outlined in the preceding paragraphs these assumptions are unlikely to hold. Some of the available FLOP/s can therefore be considered as 'lost'. For the hexahedral mesh the fraction of lost FLOP/s varies from 22.5% when $\wp = 1$ to 8.7% in the case of $\wp = 4$. With the exception of $\wp = 1$ the fraction of lost FLOP/s are a few percent higher for the mixed mesh. This is understandable given the additional complexities associated with mixed mesh partitioning and can likely be improved upon by switching to order-dependent element weighting factors.

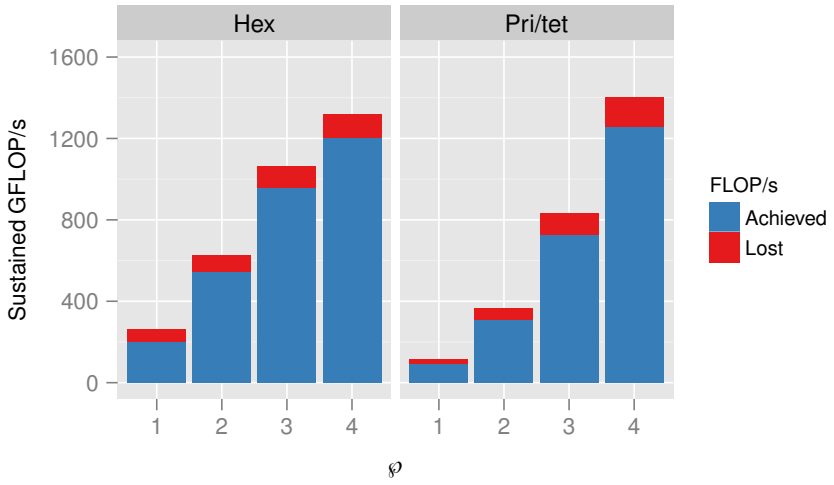


Figure 6.18. Sustained performance of PyFR on the multi-node heterogeneous system for each mesh with $\wp = 1,2,3,4$. Lost FLOP/s represent the difference between the achieved FLOP/s and the sum of the E5-2697 (C/OpenMP), K40c (CUDA), and W9100 (OpenCL A) bars in Figure 6.17.

6.6 Scalability

The scalability of PyFR v1.0.0 has been evaluated on the Piz Daint supercomputer [92]. Housed at the Swiss National Supercomputing Centre (CSCS) it is based around the Cray X30 platform and has 5 272 NVIDIA K20X GPUs. Each GPU has a theoretical peak of 1 311 GFLOP/s for a total of 7.8 PFLOP/s. The raw memory capacity of each GPU is 6 GiB however this decreases to ~5.25 GiB when ECC is enabled.

When examining the scalability of a code there are two commonly used metrics. The first of these is weak scalability in which the size of the target problem is increased in proportion to the number of ranks N. A code is said to have perfect weak scalability if the runtime remains unchanged as more ranks are added. The second metric is strong scalability wherein the problem size is fixed and the speedup compared to a starting number of ranks, N_0 is assessed. Perfect strong scalability implies that the runtime scales as N_0/N.

Table 6.9. Weak scalability of PyFR at $\wp = 4$.

# K20X	2	4	8	40	80	160	2000
Runtime	1.00	1.00	1.02	1.03	1.07	1.05	1.05
TFLOP/s	0.70	1.39	2.74	13.52	26.18	53.09	1332.52

To evaluate the scalability of PyFR a NACA 0021 aerofoil was meshed in two dimensions using 51 632 unstructured quadrilateral elements. The grid was then extruded to give a one layer $N_L = 1$ hexahedral grid. When in double precision with the Navier–Stokes solver in PyFR at $\wp = 4$ using full anti-aliasing—consisting of a 216 point rule in the volume and a 36 point rule on each face—this results in a working set of ~6.4 GiB. For the purposes of performance evaluation the problem can be scaled arbitrarily by increasing the number of layers N_L in the extrusion. Before any simulations can be run it is necessary to first partition the domain into N pieces. This is accomplished using METIS [90]. An important consequence of this is that the metrics being measured are a function of both the inherit scalability of PyFR and of the quality of the domain decomposition.

Weak scalability. Starting with two K20X GPUs and a single layer, for a working set of ~3.2 GiB/GPU, the weak scalability of PyFR was evaluated up to $N = 2\,000$. The resulting runtimes, normalised to that of the $N = 2$ case, are tabulated in Table 6.9. In the case of $N = 2\,000$ the simulation is observed to consist of 32×10^9 degrees of freedom with a total working set of ~6.25 TiB. The resulting sustained performance of 1.3 PFLOP/s represents 50.8% of the theoretical FLOP/s. This is extremely impressive for a high-order code running on an automatically partitioned unstructured grid.

Strong scalability. Starting with 50 K20X GPUs and forty layers, for a net working set of ~256 GiB, the strong scalability of PyFR was evaluated up to $N = 400$. The resulting speedups compared to the initial $N = 50$ case are tabulated in Table 6.10. From the table it is observed that an eight-fold increase in GPUs results in a speed up

Table 6.10. Strong scalability of PyFR at $p = 4$.

# K20X	50	100	200	400
Speedup	1.0	1.96	3.67	6.26
TFLOP/S	33.64	65.87	123.46	210.66

of 6.26. Although this is not perfect it is important to note that in this case the working set of each GPU is just ~640 MiB.

Chapter 7

Conclusion

A formulation of the FR approach has been developed for solving non-linear advection diffusion type problems on mixed curvilinear grids. It has also been demonstrated how the majority of operations within this formulation can be cast as large matrix-matrix multiplications. Furthermore, a methodology has been presented for automatically determining the maximum stable step size for a simulation. Techniques for mitigating and controlling the impact of aliasing driven instabilities have also been investigated.

As part of this a methodology for identifying symmetric quadrature rules on a variety of domains in two and three dimensions was presented. Using this methodology a set of rules tuned towards the requirements of finite element methods, including anti-aliased FR, were presented. Many of these rules appear to be new and represent and improvement over those tabulated in the literature. The impact of solution point placement on the nonlinear stability of FR schemes has also been studied extensively. Theoretical results confirming the optimal nature of Gauss-Legendre points were presented. A new class of Lebesgue and truncation optimised solution points were also derived for triangular elements and shown to represent an improvement over existing point sets.

PyFR, an open source Python based framework for solving the Euler and compressible Navier–Stokes equations on mixed unstructured grids, has also been presented. The structure and ethos of PyFR has been explained including the approaches taken to support multiple hardware platforms. It is shown how runtime code generation can be used to improve both the performance and portability of the code. Extensive validation of PyFR has also been performed. Spatial super accuracy is demonstrated when solving the two dimensional Euler equations along with the expected orders of accuracy for the Couette flow problem on a range of grids in two and three dimensions. The long

time dynamics of flow over a cylinder at $Re = 3\,900$ were also assessed with PyFR successfully resolving both the L and H modes. Results demonstrating the performance portability of PyFR across a range of hardware platforms were also presented. The heterogeneous capabilities of PyFR were also demonstrated. The scalability of PyFR has been demonstrated in the weak sense up to 2 000 NVIDIA K20X GPUs when solving the three dimensional Navier–Stokes equations around an extruded NACA 0021 aerofoil. On an unstructured grid sustained performance in excess of 1.3 PFLOP/s is observed.

Appendix A

Approximate Riemann Solvers

In the following section u_L and u_R are taken to be the two discontinuous solution states at an interface and $\hat{\mathbf{n}}_L$ to be the normal vector associated with the first state. For convenience $\mathbf{f}_L^{(\text{inv})} = \mathbf{f}^{(\text{inv})}(u_L)$, and $\mathbf{f}_R^{(\text{inv})} = \mathbf{f}^{(\text{inv})}(u_R)$ with inviscid fluxes being prescribed by (2.44).

A.1 Rusanov

Also known as the local Lax-Friedrichs method a Rusanov type Riemann solver imposes inviscid numerical interface fluxes according to

$$\mathfrak{F}^{(\text{inv})} = \frac{\hat{\mathbf{n}}_L}{2} \cdot \left\{ \mathbf{f}_L^{(\text{inv})} + \mathbf{f}_R^{(\text{inv})} \right\} + \frac{s}{2}(u_L - u_R), \tag{A.1}$$

where s is an estimate of the maximum wave speed

$$s = \sqrt{\frac{\gamma(p_L + p_R)}{\rho_L + \rho_R}} + \frac{1}{2}\left| \hat{\mathbf{n}}_L \cdot (\mathbf{v}_L + \mathbf{v}_R) \right|. \tag{A.2}$$

Appendix B

Boundary Conditions

To incorporate boundary conditions into the FR approach a set of boundary interface types $b \in \mathcal{B}$ are introduced. At a boundary interface there is only a single flux point: that which belongs to the element whose edge/face is on the boundary. Associated with each boundary type are a pair of functions $\mathfrak{C}_\alpha^{(b)}(u_L)$ and $\mathfrak{F}_\alpha^{(b)}(u_L, \mathbf{q}_L, \hat{\mathbf{n}}_L)$ where u_L, $\mathbf{q}_L$, and $\hat{\mathbf{n}}_L$ are the solution, solution gradient and unit normals at the relevant flux point. These functions prescribe the common solutions and normal fluxes, respectively.

Instead of directly imposing solutions and normal fluxes it is oftentimes more convenient for a boundary to instead provide ghost states. In its simplest formulation $\mathfrak{C}_\alpha^{(b)} = \mathfrak{C}_\alpha(u_L, \mathfrak{B}^{(b)}u_L)$ and $\mathfrak{F}_\alpha^{(b)} = \mathfrak{F}_\alpha(u_L, \mathfrak{B}^{(b)}u_L, \mathbf{q}_L, \mathfrak{B}^{(b)}\mathbf{q}_L, \hat{\mathbf{n}}_L)$ where $\mathfrak{B}^{(b)}u_L$ is the ghost solution state and $\mathfrak{B}^{(b)}\mathbf{q}_L$ is the ghost solution gradient. It is straightforward to extend this prescription to allow for the provisioning of different ghost solution states for $\mathfrak{C}_\alpha$ and $\mathfrak{F}_\alpha$ and to permit $\mathfrak{B}^{(b)}\mathbf{q}_L$ to be a function of u_L in addition to $\mathbf{q}_L$.

B.1 Supersonic Inflow

The supersonic inflow condition is parameterised by a free-stream density ρ_f, velocity $\mathbf{v}_f$, and pressure p_f.

$$\mathfrak{B}^{(\text{inv})}u_L = \mathfrak{B}^{(\text{ldg})}u_L = \begin{Bmatrix} \rho_f \\ \rho_f \mathbf{v}_f \\ p_f/(\gamma - 1) + \rho_f \|\mathbf{v}_f\|^2/2 \end{Bmatrix}, \tag{B.1}$$

$$\mathfrak{B}^{(\text{ldg})}\mathbf{q}_L = 0. \tag{B.2}$$

B.2 Subsonic Outflow

Subsonic outflow boundaries are parameterised by a free-stream pressure p_f.

$$\mathfrak{B}^{(\text{inv})}u_L = \mathfrak{B}^{(\text{ldg})}u_L = \begin{Bmatrix} \rho_L \\ \rho_L \mathbf{v}_L \\ p_f/(\gamma-1) + \rho_L \|\mathbf{v}_L\|^2/2 \end{Bmatrix}, \tag{B.3}$$

$$\mathfrak{B}^{(\text{ldg})}\mathbf{q}_L = 0, \tag{B.4}$$

B.3 No-slip Isothermal Wall

The no-slip isothermal wall condition depends on the wall temperature $c_p T_w$ and the wall velocity $\mathbf{v}_w$. Usually $\mathbf{v}_w = 0$.

$$\mathfrak{B}^{(\text{inv})}u_L = \rho_L \begin{Bmatrix} 1 \\ 2\mathbf{v}_w - \mathbf{v}_L \\ c_p T_w/\gamma + \|2\mathbf{v}_w - \mathbf{v}_L\|^2/2 \end{Bmatrix}, \tag{B.5}$$

$$\mathfrak{B}^{(\text{ldg})}u_L = \rho_L \begin{Bmatrix} 1 \\ \mathbf{v}_w \\ c_p T_w/\gamma + \|\mathbf{v}_w\|^2/2 \end{Bmatrix}, \tag{B.6}$$

$$\mathfrak{B}^{(\text{ldg})}\mathbf{q}_L = \mathbf{q}_L. \tag{B.7}$$

B.4 Characteristic Riemann Invariant Far-Field

The characteristic Riemann invariant far-field boundary condition follows the prescription of Jameson and Baker [93] and is parameterised by a free-stream density ρ_f, velocity $\mathbf{v}_f$, and pressure p_f. At the boundary the internal and free-stream sound speeds can be computed as $c_L = \sqrt{\gamma \rho_L / p_L}$ and $c_f = \sqrt{\gamma \rho_f / p_f}$, respectively. With these the

Riemann invariants can be introduced as

$$R_L = \begin{cases} \mathbf{v_f} \cdot \hat{\mathbf{n}}_L + 2c_f/(\gamma - 1) & \text{if } |\mathbf{v}_f \cdot \hat{\mathbf{n}}_L| \geq c_f \text{ and } \mathbf{v_L} \cdot \hat{\mathbf{n}}_L \geq 0 \\ \mathbf{v_L} \cdot \hat{\mathbf{n}}_L + 2c_L/(\gamma - 1) & \text{otherwise,} \end{cases} \tag{B.8}$$

$$R_f = \begin{cases} \mathbf{v_L} \cdot \hat{\mathbf{n}}_L - 2c_L/(\gamma - 1) & \text{if } |\mathbf{v}_f \cdot \hat{\mathbf{n}}_L| \geq c_f \text{ and } \mathbf{v_L} \cdot \hat{\mathbf{n}}_L < 0 \\ \mathbf{v_f} \cdot \hat{\mathbf{n}}_L - 2c_f/(\gamma - 1) & \text{otherwise.} \end{cases} \tag{B.9}$$

Using these the density, velocity, and pressure at the boundary can be defined as

$$\rho_b^{\gamma-1} = \frac{(\gamma-1)^2(R_L - R_f)^2}{16\gamma} \begin{cases} \rho_f^\gamma / p_f & \text{if } \mathbf{v_L} \cdot \hat{\mathbf{n}}_L < 0 \\ \rho_L^\gamma / p_L & \text{otherwise,} \end{cases} \tag{B.10}$$

$$\mathbf{v}_b = \frac{\hat{\mathbf{n}}_L}{2}(R_L + R_f) \begin{cases} \mathbf{v}_f - \hat{\mathbf{n}}_L(\mathbf{v}_f \cdot \hat{\mathbf{n}}_L) & \text{if } \mathbf{v_L} \cdot \hat{\mathbf{n}}_L < 0 \\ \mathbf{v}_L - \hat{\mathbf{n}}_L(\mathbf{v}_L \cdot \hat{\mathbf{n}}_L) & \text{otherwise,} \end{cases} \tag{B.11}$$

$$p_b = \frac{(\gamma-1)^2(R_L - R_f)^2 \rho_b}{16\gamma}, \tag{B.12}$$

with the final boundary states being given by

$$\mathfrak{B}^{(\text{inv})} u_L = \mathfrak{B}^{(\text{ldg})} u_L = \begin{Bmatrix} \rho_b \\ \rho_b \mathbf{v}_b \\ p_b/(\gamma - 1) + \rho_b \|\mathbf{v}_b\|^2/2 \end{Bmatrix}, \tag{B.13}$$

$$\mathfrak{B}^{(\text{ldg})} \mathbf{q}_L = 0. \tag{B.14}$$

Bibliography

[1] PE Vincent and A Jameson. Facilitating the adoption of unstructured high-order methods amongst a wider community of fluid dynamicists.*Mathematical Modelling of Natural Phenomena* 6(03), 2011, pp. 97–140.

[2] A Jameson and K Ou. 50 years of transonic aircraft design. *Progress in Aerospace Sciences* 47(5), 2011, pp. 308–318.

[3] A Harten, B Engquist, S Osher, and SR Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, III. *Journal of Computational Physics* 71(2), 1987, pp. 231–303.

[4] XD Liu, S Osher, and T Chan. Weighted essentially non-oscillatory schemes. *Journal of computational physics* 115(1), 1994, pp. 200–212.

[5] G Karniadakis and SJ Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, 2005.

[6] P Solin, K Segeth, and I Dolezel. *Higher-order finite element methods*. CRC Press, 2003.

[7] WH Reed and TR Hill. Triangular mesh methods for the neutron transport equation. *Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory*, 1973.

[8] DA Kopriva and JH Kolias. A conservative staggered-grid Chebyshev multidomain method for compressible flows. *Journal of computational physics* 125(1), 1996, pp. 244–261.

[9] Y Sun, ZJ Wang, and Y Liu. High-order multidomain spectral difference method for the Navier–Stokes equations on unstructured hexahedral grids.*Communications in Computational Physics* 2(2), 2007, pp. 310–333.

[10] HT Huynh. A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods. *AIAA paper* 4079, 2007.

[11] JS Hesthaven and T Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Vol. 54. Springer Verlag New York, 2008.

[12] H Gao and ZJ Wang. A high-order lifting collocation penalty formulation for the Navier-Stokes equations on 2D mixed grids. *ratio* 1, 2009, p. 2.

[13] ZJ Wang and H Gao. A unifying lifting collocation penalty formulation including the discontinuous Galerkin, spectral volume/difference methods for conservation laws on mixed grids. *Journal of Computational Physics* 228(21), 2009, pp. 8161–8186.

[14] M Yu and ZJ Wang. On the connection between the correction and weighting functions in the correction procedure via reconstruction method. *Journal of Scientific Computing* 54(1), 2013, pp. 227–244.

[15] Y Allaneau and A Jameson. Connections between the filtered discontinuous Galerkin method and the flux reconstruction approach to high order discretizations. *Computer Methods in Applied Mechanics and Engineering* 200(49), 2011, pp. 3628–3636.

[16] DA Kopriva. A staggered-grid multidomain spectral method for the compressible Navier–Stokes equations. *Journal of Computational Physics* 143(1), 1998, pp. 125–158.

[17] E Hairer, SP Nørsett, and G Wanner. *Solving Ordinary Differential Equations I.* 2nd ed. Springer-Verlag, 1993.

[18] WH Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[19] MH Carpenter and CA Kennedy. *Fourth-Order 2N-Storage Runge–Kutta Schemes*. 1994.

[20] CA Kennedy, MH Carpenter, and RM Lewis. Low-storage, explicit Runge–Kutta schemes for the compressible Navier–Stokes equations. *Applied numerical mathematics* 35(3), 2000, pp. 177–219.

[21] JC Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, 2008. ISBN: 9780470753767.

[22] E Hairer and G Wanner. *Solving Ordinary Differential Equations II*. 2nd ed. Springer-Verlag, 1996.

[23] PE Vincent, P Castonguay, and A Jameson. Insights from von Neumann analysis of high-order flux reconstruction schemes. *Journal of Computational Physics* 230(22), 2011, pp. 8134–8154.

[24] PE Vincent, P Castonguay, and A Jameson. A new class of high-order energy stable flux reconstruction schemes. *Journal of Scientific Computing* 47(1), 2011, pp. 50–72.

[25] A Jameson. A proof of the stability of the spectral difference method for all orders of accuracy. *Journal of Scientific Computing* 45(1-3), 2010, pp. 348–358.

[26] P Castonguay, DM Williams, PE Vincent, and A Jameson. Energy Stable Flux Reconstruction Schemes for Advection-Diffusion Problems. *Computer Methods in Applied Mechanics and Engineering*, 2013.

[27] P Castonguay, PE Vincent, and A Jameson. A new class of high-order energy stable flux reconstruction schemes for triangular elements.*Journal of Scientific Computing* 51(1), 2012, pp. 224–256.

[28] DM Williams, P Castonguay, PE Vincent, and A Jameson. Energy stable flux reconstruction schemes for advection-diffusion problems on triangles. *Journal of Computational Physics*, 2013.

[29] HT Huynh. High-order methods including discontinuous Galerkin by reconstructions on triangular meshes. *AIAA Paper* 44, 2011.

[30] DM Williams and A Jameson. Energy Stable Flux Reconstruction Schemes for Advection-Diffusion Problems on Tetrahedra. *Journal of Scientific Computing*, 2013, pp. 1–39.

[31] PE Vincent, AM Farrington, FD Witherden, and A Jameson. An extended range of stable-symmetric-conservative flux reconstruction correction functions. *Computer Methods in Applied Mechanics and Engineering* 296, 2015, pp. 248–272.

[32] P Castonguay, PE Vincent, and A Jameson. A new class of high-order energy stable flux reconstruction schemes for triangular elements. *Journal of Scientific Computing*, 2011.

[33] A Jameson, PE Vincent, and P Castonguay. On the non-linear stability of flux reconstruction schemes. *Journal of Scientific Computing* 50(2), 2011, pp. 434–445.

[34] P Castonguay, PE Vincent, and A Jameson. Application of high-order energy stable flux reconstruction schemes to the Euler equations. *AIAA paper* 686, 2011.

[35] DM Williams and A Jameson. Nodal Points and the Nonlinear Stability of High-Order Methods for Unsteady Flow Problems on Tetrahedral Meshes. *AIAA paper* 2830, 2013.

[36] DM Williams, L Shunn, and A Jameson. Symmetric quadrature rules for simplexes based on sphere close packed lattice arrangements. *Journal of Computational and Applied Mathematics* 266, 2014, pp. 18–38.

[37] FD Witherden and PE Vincent. An analysis of solution point coordinates for flux reconstruction schemes on triangular elements. *Journal of Scientific Computing* 61(2), 2014, pp. 398–423.

[38] RM Kirby and G Karniadakis. De-aliasing on non-uniform grids: algorithms and applications. *Journal of Computational Physics* 191(1), 2003, pp. 249–264.

[39] RM Kirby and SJ Sherwin. Aliasing errors due to quadratic nonlinearities on triangular spectral/hp element discretisations. *Journal of engineering mathematics* 56(3), 2006, pp. 273–288.

[40] SA Orszag. On the elimination of aliasing in finite-difference schemes by filtering high-wavenumber components. *Journal of the Atmospheric sciences* 28(6), 1971, pp. 1074–1074.

[41] EF Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer, 2009.

[42] JN Lyness and D Jespersen. Moderate degree symmetric quadrature rules for the triangle. *IMA Journal of Applied Mathematics* 15(1), 1975, pp. 19–32.

[43] DA Dunavant. High degree efficient symmetrical Gaussian quadrature rules for the triangle. *International journal for numerical methods in engineering* 21(6), 1985, pp. 1129–1148.

[44] JN Lyness and R Cools. A survey of numerical cubature over triangles. *Proceedings of Symposia in Applied Mathematics*. Vol. 48. 1994, pp. 127–150.

[45] JS Savage and AF Peterson. Quadrature rules for numerical integration over triangles and tetrahedra. *Antennas and Propagation Magazine, IEEE* 38(3), 1996, pp. 100–102.

[46] S Wandzurat and H Xiao. Symmetric quadrature rules on a triangle.*Computers & Mathematics with Applications* 45(12), 2003, pp. 1829–1840.

[47] L Zhang, T Cui, and H Liu. A set of symmetric quadrature rules on triangles and tetrahedra. *J. Comput. Math* 27(1), 2009, pp. 89–96.

[48] MA Taylor, BA Wingate, and LP Bos. Several new quadrature formulas for polynomial integration in the triangle. *ArXiv Mathematics e-prints*, 2005.

[49] H Xiao and Z Gimbutas. A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions. *Computers & mathematics with applications* 59(2), 2010, pp. 663–676.

[50] DA Dunavant. Economical symmetrical quadrature rules for complete polynomials over a square domain. *International journal for numerical methods in engineering* 21(10), 1985, pp. 1777–1784.

[51] R Cools and A Haegemans. Another step forward in searching for cubature formulae with a minimal number of knots for the square. *Computing* 40(2), 1988, pp. 139–146.

[52] L Shunn and F Ham. Symmetric quadrature rules for tetrahedra based on a cubic close-packed lattice arrangement. *Journal of Computational and Applied Mathematics* 236(17), 2012, pp. 4348–4364.

[53] P Keast. Moderate-degree tetrahedral quadrature formulas. *Computer Methods in Applied Mechanics and Engineering* 55(3), 1986, pp. 339–348.

[54] EJ Kubatko, BA Yeager, and AL Maggi. New computationally efficient quadrature formulas for triangular prism elements. *Computers & Fluids* 73, 2013, pp. 187–201.

[55] EJ Kubatko, BA Yeager, and AL Maggi. New computationally efficient quadrature formulas for pyramidal elements. *Finite Elements in Analysis and Design* 65, 2013, pp. 63–75.

[56] AH Stroud. *Approximate calculation of multiple integrals*. Prentice-Hall, 1971.

[57] DA Dunavant. Efficient symmetrical cubature rules for complete polynomials of high degree over the unit cube. *International journal for numerical methods in engineering* 23(3), 1986, pp. 397–407.

[58] R Cools and KJ Kim. Rotation invariant cubature formulas over the n-dimensional unit cube. *Journal of computational and applied mathematics* 132(1), 2001, pp. 15–32.

[59] FWJ Olver. *NIST handbook of mathematical functions*. Cambridge University Press, 2010.

[60] G Guennebaud, B Jacob, et al. *Eigen v3*. 2010. http://eigen.tuxfamily.org.

[61] L Fousse, G Hanrot, V Lefèvre, P Pélissier, and P Zimmermann. MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding. *ACM Transactions on Mathematical Software* 33(2), 2007, 13:1–13:15.

[62] L Bos. On certain configurations of points in R^n which are unisolvent for polynomial interpolation. *Journal of approximation theory* 64(3), 1991, pp. 271–280.

[63] TJ Rivlin. *An introduction to the approximation of functions*. Dover, 2003.

[64] T Warburton. An explicit construction of interpolation nodes on the simplex. *Journal of engineering mathematics* 56(3), 2006, pp. 247–262.

[65] MA Taylor, BA Wingate, and RE Vincent. An algorithm for computing Fekete points in the triangle. *SIAM Journal on Numerical Analysis* 38(5), 2000, pp. 1707–1720.

[66] Q Chen and I Babuška. The optimal symmetrical points for polynomial interpolation of real functions in the tetrahedron. *Computer methods in applied mechanics and engineering* 137(1), 1996, pp. 89–94.

[67] H Luo and C Pozrikidis. A Lobatto interpolation grid in the tetrahedron. *IMA journal of applied mathematics*, 2006.

[68] J Chan and T Warburton. A Comparison of High Order Interpolation Nodes for the Pyramid. *arXiv preprint arXiv:1412.4138*, 2014.

[69] F Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*. 2013. http://mpmath.org/.

[70] M Bayer. *Mako: Templates for Python*. 2013. http://www.makotemplates.org/.

[71] A Klöckner, N Pinto, Y Lee, B Catanzaro, P Ivanov, and A Fasih. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. *Parallel Comput.* 38(3), 2012, pp. 157–174. ISSN: 0167-8191.

[72] L Dalcin. *mpi4py: MPI for Python*. 2013. https://bitbucket.org/mpi4py.

[73] A Collette. *Python and HDF5*. O'Reilly Media, 2013.

[74] BC Vermeire and S Nadarajah. Adaptive IMEX time-stepping for ILES using the correction procedure via reconstruction scheme. *AIAA paper* 2687, 2013.

[75] BC Vermeire and S Nadarajah. Adaptive IMEX schemes for high-order unstructured methods. *Journal of Computational Physics* 280, 2015, pp. 261–286.

[76] C Norberg. LDV measurements in the near wake of a circular cylinder.*International Journal for Numerical Methods in Fluids* 28(9), 1998, pp. 1281–1302.

[77] X Ma, GS Karamanos, and GE Karniadakis. Dynamics and low-dimensionality of a turbulent near wake. *Journal of Fluid Mechanics* 310, 2000, pp. 29–65.

[78] M Breuer. Large eddy simulation of the subcritical flow past a circular cylinder. *International Journal for Numerical Methods in Fluids* 28(9), 1998, pp. 1281–1302.

[79] AG Kravchenko and P Moin. Numerical studies of flow over a circular cylinder at $Re_D = 3\,900$. *Physics of Fluids* 12, 2000, pp. 403–417.

[80] P Parnaudeau, J Carlier, D Heitz, and E Lamballais. Experimental and numerical studies of the flow over a circular cylinder at Reynolds number 3900. *Physics of Fluids* 20(8), 2008.

[81] A Roshko. On the development of turbulent wakes from vortex streets. *Technical Report No. NACA TR 1191, California Institute of Technology*, 1953.

[82] MS Bloor. The transition to turbulence in the wake of a circular cylinder. *Journal of Fluid Mechanics* 19, 1964, pp. 290–304.

[83] CHK Williamson. The existence of two stages in the transition to three dimensionality of a cylinder wake. *Physics of Fluids* 31, 1988, pp. 3165–3168.

[84] CHK Williamson. Vortex dynamics in the cylinder wake. *Annual Review of Fluid Mechanics* 28, 1996, pp. 477–539.

[85] FD Witherden, BC Vermeire, and PE Vincent. Heterogeneous computing on mixed unstructured grids with PyFR. *Computers & Fluids* 120, 2015, pp. 173–186.

[86] O Lehmkuhl, I Rodriguez, R Borrell, and A Oliva. Low-frequency unsteadiness in the vortex formation region of a circular cylinder. *Physics of Fluids* 25(8), 2013, pp. 3165–3168.

[87] BC Vermeire, JS Cagnone, and S Nadarajah. ILES using the correction procedure via reconstruction scheme. *AIAA paper* 1001, 2013.

[88] BC Vermeire, S Nadarajah, and PG Tucker. Canonical test cases for high-order unstructured implicit large eddy simulation. *AIAA paper* 0935, 2014.

[89] FD Witherden, AM Farrington, and PE Vincent. PyFR: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications* 185(11), 2014, pp. 3028–3040.

[90] G Karypis and V Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 20(1), 1998, pp. 359–392.

[91] F Pellegrini and J Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. *High-Performance Computing and Networking*. Springer. 1996, pp. 493–498.

[92] PE Vincent, FD Witherden, AM Farrington, G Ntemos, BC Vermeire, JS Park, and AS Iyer. PyFR: Next-Generation High-Order Computational Fluid Dynamics on Many-Core Hardware. *AIAA paper* 3050, 2015.

[93] A Jameson and T Baker. Solution of the Euler equations for complex configurations. *AIAA Paper* 1929, 1983.

Colophon

The original source for this document was typeset by the author in LaTeX 2_ε using the KOMA-Script bundle. Diagrams and illustrations were created using the Ti*k*Z package. Graphs were generated in R using the ggplot2 package. All of the source was written in GNU Emacs with the AUCTeX package. The final document was generated using LuaTeX and optimised using pdfsizeopt.

The title and captioning font is URW-Garamond while the main body font is Times at 11pt. Sans-serif elements are typeset in Helvetica.