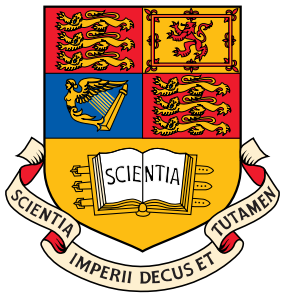
From h to p Efficiently: Optimising the Implementation of Spectral/hp Element Methods

by

Peter Edward Julia Vos



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

This thesis is submitted for the degree of Doctor of Philosophy of the University of London

2011

Declaration

This is to certify that the work presented in this thesis has been carried out at Imperial College London, and has not been previously submitted to any other university or technical institution for a degree or award. The thesis comprises only my original work, except where due acknowledgement is made in the text.

Peter Vos

Abstract

Various aspects that may help to enhance the implementation of the spectral/*hp* element method have been considered.

A first challenge encountered is to implement the method and the corresponding algorithms in a digestible, generic and coherent manner. Therefore, we first of all demonstrate how the mathematical structure of a spectral/*hp* element discretisation can be encapsulated in an object-oriented environment, leading to a generic and flexible spectral/*hp* software library. Secondly, we present a generic framework for time-stepping partial differential equations. Based upon the unifying concept of *General Linear Methods*, we have designed an object-oriented framework that allows the user to apply a broad range of time-stepping schemes in a unified fashion.

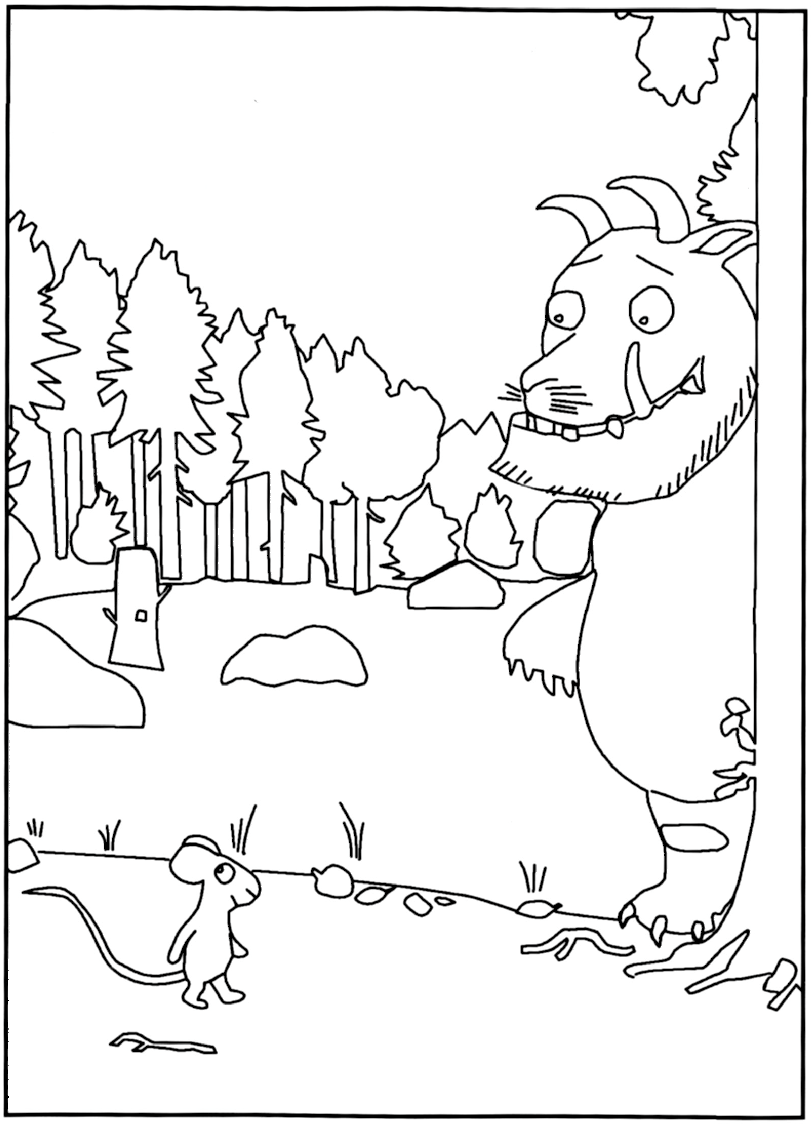
The spectral/*hp* element method can be considered as bridging the gap between the – traditionally low-order – finite element method on one side and spectral methods on the other side. Consequently, a second challenge which arises in implementing the spectral/*hp* element methods is to design algorithms that perform efficiently for both low- and high-order spectral/*hp* discretisations, as well as discretisations in the intermediate regime. In this thesis, we describe how the judicious use of different implementation strategies for the evaluation of spectral/*hp* operators can be employed to achieve high efficiency across a wide range of polynomial orders. Furthermore, we explain how the multi-level static condensation technique can be applied as an efficient direct solution technique for solving linear systems that arise in the spectral/*hp* element method.

Finally, based upon such an efficient implementation of the spectral/*hp* element method, we analyse which spectral/*hp* discretisation (that is, which specific combination of mesh size h and polynomial order P) minimises the computational cost to solve an elliptic problem up to a predefined level of accuracy. We investigate this question for a set of both smooth and non-smooth problems.

Acknowledgements

I would like to express my deepest gratitude to my supervisors Professor Spencer Sherwin and Professor Mike Kirby. I would like to thank both of them not only for their support and valuable contributions concerning the content of this thesis, but also for their guidance during my first steps in academia. A special thanks to Spencer for offering me this opportunity and for the confidence he put in me during the last four years. I am also indebted to Dr. ir. Marc Gerritsma, who through his earlier MSc supervision, has had a continued influence on my work as a scientist. Thank you also to all the people of the Nektar++ project who helped with the example section of the time-stepping chapter. In particular, I would like to thank Claes Eskilsson for his generous help and his moral support.

I would also would like to thank my friends for their friendship, but most of all, I would like to thank my wife Tine. In many ways, this work can be considered as a joint effort and, therefore, much of the credit for the completion of this thesis goes to her. Thank you my love, I am glad we have finally made it ...



for Jules and Hannah

who all the time were being told their father was writing a book

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The spectral/*hp* element method combines the geometric flexibility of classical *h*-type finite element techniques with the desirable resolution properties of spectral methods. In this approach a polynomial expansion of order P is applied to every elemental domain of a coarse finite element type mesh. These techniques have been applied in many fundamental studies of fluid mechanics (Sherwin & Karniadakis 1996) and more recently have gained greater popularity in the modelling of wave-based phenomena such as computational electromagnetics (Hesthaven & Warburton 2002) and shallow water problems (Bernard, Remacle, Combien, Legat, & Hillewaert 2009) – particularly when applied within a Discontinuous Galerkin formulation. Although there are many references on the spectral/*hp* element method today, including the textbooks of (Karniadakis & Sherwin 2005; Deville, Fischer, & Mund 2002; Hesthaven & Warburton 2008), few references exist that cover in detail the implementational aspects of the method. This thesis makes an attempt to fill this gap by investigating some of the fundamental questions related to the implementation of the spectral/*hp* element method.

There are at least two major challenges which arise in developing an efficient implementation of a spectral/*hp* element discretisation:

- implementing the mathematical structure of the technique in a digestible, generic and coherent manner, and

- designing and implementing the numerical methods and data structures in a matter so that both high- and low-order discretisations can be efficiently applied.

The present work intends to reflect on both these objectives. The efforts made in the context of this thesis have contributed to the ongoing development of the Nektar++ project (Kirby & Sherwin 2006). Nektar++ is an open source software library which is currently being collaboratively developed between engineers at Imperial College London with computer scientists at the University of Utah. The first objective in mind, the C++ programming language has been adopted within this collaboration as the object-oriented coding environment that provides the necessary data structuring and algorithms that appropriately emulate the mathematical construction of a spectral/*hp* discretisation. We believe that this type of structuring will facilitate a wider class of developers in applying spectral/*hp* discretisations to scientific and engineering applications without having to be overly familiar with the fine details of the implementations. In this work, we will present how we have decided to encapsulate spectral/*hp* element discretisations within the Nektar++ project and we will also demonstrate how an object-oriented environment allows us to design generic algorithms by presenting the framework we have developed to time-integrate partial differential equations (PDEs) in a unified fashion. Next to these considerations above, which are an important issue in architecting the library and developing generic algorithms, a large part of this thesis is dedicated to the second objective regarding efficiency and computational performance.

In order to design algorithms which are efficient for both low- and high-order spectral/*hp* discretisations, it is important to clearly define what we mean with low- and high-order. The spectral/*hp* element method can be considered as bridging the gap between the high-order end of the traditional finite element method and low-order end of conventional spectral methods. However, the concept of high- and low-order discretisations can mean very different things to these different communities. For example, the seminal works by Zienkiewicz & Taylor (Zienkiewicz & Taylor 1989) and Hughes (Hughes 1987) list examples of elemental expansions only up to third or possibly fourth-order, implying that these orders are considered to be high-order for

the traditional h-type finite element community. In contrast the text books of the spectral/hp element community (Szabó & Babuška 1991; Karniadakis & Sherwin 2005; Deville, Fischer, & Mund 2002; Hesthaven & Warburton 2008) typically show examples of problems ranging from a low-order bound of minimally fourth-order up to anything ranging from 10^{th}-order to 15^{th}-order polynomial expansions. On the other end of the spectrum, practitioners of global (Fourier-based) spectral methods (Gottlieb & Orszag 1977) would probably consider a 16^{th}-order global expansion to be relatively low-order approximation.

One could wonder whether these different definitions of low- and high-order are just inherent to the tradition and lore of each of the communities or whether there are more practical reasons for this distinct interpretation. Proponents of lower-order methods might highlight that some problems of practical interest are so geometrically complex that one cannot computationally afford to use high-order techniques on the massive meshes required to capture the geometry. Alternatively, proponents of high-order methods highlight that if the problem of interest can be captured on a computational domain at reasonable cost then using high-order approximations for sufficiently *smooth* solutions will provide a higher accuracy for a given computational cost. If one however probes even further it also becomes evident that the different communities choose to implement their algorithms in different manners. For example the standard h-type finite element community will typically uses techniques such as sparse matrix storage formats (where only the non-zero entries of a global matrix are stored) to represent a global operator. In contrast the spectral/hp element community acknowledges that for higher polynomial expansions more closely coupled computational work takes place at the individual elemental level and this leads to the use of elemental operators rather than global matrix operators. In addition the global spectral method community often make use of the tensor-product approximations where products of one-dimensional rules for integration and differentiation can be applied. From the results in this thesis, it will appear that each of the different implementation strategies will perform poorly when applied outside the aforementioned polynomial regimes typically adopted by the different communities, hinting that the traditional views of low- and high-order may be have been strengthened by

these practical barriers.

In this thesis, we are therefore lead to ask when we should adopt these different implementation strategies if we would like to go *from* h *to* p *efficiently*, that is, if we are to allow the order of our spatial approximations to vary from low-order ($P = 1$) up to high-order (say $P = 15$)? We note that analytic estimates of computational work in this polynomial regime are difficult if not impossible to establish since the computational effort is highly dependent on hard to predict hardware characteristics such as memory management and caching effect as well as optimised linear algebra packages such as BLAS (Dongarra, Du Croz, Hammarling, Hanson, & Duff 1988) and LAPACK (Anderson, Bai, Bischof, Blackford, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney, & Sorensen 1999). We therefore will mainly follow a computational approach to assess the efficiency of the different implementation strategies. The support of various implementation strategies within a spectral/*hp* code will allow the user to cross the community dependent barriers of low- and high-order in an efficient way such as the aforementioned example of $P = 4$ (the high-order limit for traditional finite elements and the low-order limit for spectral/*hp* elements). This surrounding polynomial regime ($2 < P < 6$) is however potentially an optimal/desirable range for applications where the mesh resolution is such that increasing polynomial order leads to the onset of rapid/spectral convergence. This level of resolution might be necessary to capture, for example, a complex geometry. The benefit of intermediate polynomial resolution will however only be observed if one can efficiently implement these polynomial discretisations.

Finally, we can also question whether it is sufficient to know which implementation strategy is optimal in terms of CPU time for a specific polynomial order discretisation? Probably a more pertinent question is consider one is *given the most efficient implementation, what is the best spectral/*hp *discretisation to obtain a fixed error for a minimal computational cost?* Since computational cost is impacted by different discretisation methodologies such as element size h, polynomial order P, adaptive refinement r (Remacle, Flaherty, & Shepard 2003) or even the continuity of the approximation k (Hughes, Cottrell, & Bazilevs 2005), there are clearly many factors to consider. To help reduce this extensive parameter space, in this work we

will restrict ourselves to just h and P refinements leading to the question: which specific combination of mesh-size h and polynomial order P requires the minimal computational cost (i.e. run-time) to solve a problem up to a predefined accuracy? We will investigate this question for a set of both smooth and non-smooth elliptic problems.

1.1 Motivation and structure of the thesis

As mentioned earlier in this chapter, this thesis covers different aspects that may contribute to an optimal implementation of the spectral/*hp* element method. In order to introduce the various aspects that have been studied, let us consider the scalar advection-diffusion equation, which can be seen as the driving problem throughout this thesis, defined as

$$\frac{\partial u}{\partial t} + \nabla \cdot \boldsymbol{F}(u) = \nabla^2 u. \tag{1.1}$$

For completeness, this equation should be subject to appropriate boundary and initial conditions, see also Section 4.3. In the solution procedure of this partial differential equation, we have subsequently encountered the following issues and questions listed below, which we believe matter from an implementational point of view. The thesis is structured to follow a similar pattern.

How to encapsulate the fundamental concepts related to the spatial discretisation? A first step in numerically solving the advection-diffusion equation is selecting the spatial discretisation technique, which for the scope of this thesis will be the spectral/*hp* element discretisation. Therefore in Chapter 2, we start with introducing the fundamental concepts related to the spectral/*hp* element method. In Chapter 3, we explain how these concepts can be encapsulated in a generic code environment that allows for a sufficient level of both flexibility and performance.

How to apply the time-discretisation in a generic and efficient way? Time-stepping algorithms and their implementations are, next to the spatial discretisation technique, a second important component within the solution of time-

dependent partial differential equations such as the advection-diffusion equation. Although the Method of Lines paradigm (Schiesser 1991) introduces a high level of flexibility into the choice and application of time-stepping schemes, many practitioners often feel constrained after their initial implementation of Euler-Forward to one of the two families of schemes which claim Euler-Forward as its parent method: multi-step (e.g. Adams-Bashforth and Adams-Moulton methods) or multi-stage (e.g. Runga-Kutta methods). In Chapter 4 we present a generic framework – both in terms of algorithms and implementations – that allows the user to almost seamlessly switch between almost the entire range of traditional time-stepping methods without compromising the computational performance. We will show that in case of the advection-diffusion equation, the user will only have to provide the framework with three different routines that solve the following steady problems using the selected spatial discretisation technique:

- An elliptic Helmholtz solver which numerically solves the steady Helmholtz equation

$$u - \lambda \nabla^2 u = f, \tag{1.2}$$

  subject to the imposed boundary conditions.

- A projection operator which performs the discrete L^2 projection of a function upon the discrete solution space.

- A function that evaluates the advection term $\nabla \cdot \boldsymbol{F}(u)$ according to the spatial discretisation technique.

When adopting the spectral/*hp* element method, solving the three problems above typically require two types of operations: the evaluation of spectral/*hp* element operators and the solution of linear systems. For example, the discretisation of the projection operator above leads to the discrete system

$$\boldsymbol{M}\hat{\boldsymbol{u}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{f}. \tag{1.3}$$

where $\boldsymbol{M}$ and $\boldsymbol{B}^\top \boldsymbol{W}$ respectively are the global mass matrix and the inner product operator due to the spectral/*hp* discretisation (see also Section 2.2.4). Optimising

the implementation of such discrete systems to be solved leads to the following two questions.

How to optimise the evaluation of the spectral/*hp* element operators The inner product in the discrete system above is an example of such a spectral/*hp* element operator. In Chapter 5, we explain how the judicious use of different implementation strategies can be employed to efficiently evaluate such *forward* operators for a wide range of polynomial orders. Therefore, we first introduce and discuss the three different implementation strategies using either global matrices, local matrices or the sum-factorisation technique. We then provide theoretical cost estimates for each strategy. Next we investigate the effect of the different strategies on the actual run-time by a set of computational tests and analyse which strategy should be selected depending on the polynomial order.

How to optimise the solution of the linear system? In Chapter 6, we will investigate different strategies for solving linear systems such as the one due to the global mass matrix system in the example above. We will not consider iterative solution strategies but we will restrict ourselves to *direct* solution methods. Therefore, we investigate the concept of multi-level static condensation by exploring both the top-down and the bottom-up variant. In addition, we will identify the optimal strategies by comparing both methods with the more traditional approach of bandwidth minimisation, both from a theoretical point as from a computational point of view.

What is the best *hp*-discretisation at which to run your code? Answers to the four questions above will provide some of the essential building blocks for an efficient implementation of the spectral/*hp* element method. Once these building blocks are available, it is also possible to consider the following question: given the most efficient implementation, which is the optimal combination of mesh resolution and polynomial order that requires the minimal run-time to approximate the exact solution up to predefined accuracy? In Chapter 7, we will make a first attempt to

answer this question. However, we will not consider the (time-dependent) advection-diffusion equation for this purpose, but we will limit ourselves to solving the (steady) Helmholtz equation for four different test-problems.

Finally, in Chapter 8 we will summarise the various findings, draw the relevant conclusion and finish with some recommendations for future research.

1.2 Assumptions

In this thesis, we will restrict ourselves to the traditional C^0 continuous Galerkin formulation of the spectral/*hp* element method. Furthermore, we only consider *two-dimensional* spectral/*hp* element discretisations and we assume the expansion bases to be tensorial, i.e. the two-dimensional basis functions can be constructed as a tensor product of one-dimensional basis functions (both for quadrilaterals and triangles). In addition, we assume the expansion order to be the same in both the coordinate directions.

The solution of time-dependent partial differential equations often involve the repeated application of matrix problems such as those listed in the advection-diffusion example above. As a result, we only look to optimise the *evaluation* of the matrix operators associated to the spectral/*hp* element operators (Chapter 5) or to the linear system (Chapter 6), thereby neglecting any matrix construction time (something that may matter for the steady differential equations that e.g. arise in structural mechanics). Furthermore, it should be noted that we define the optimal or most efficient implementation as the implementation which leads to the minimal run-time. Although other aspects such as the required memory may also contribute to the definition of the optimal implementation, we will associate the word *optimal* in this thesis primarily to the actual measured run-time, thereby neglecting any memory constraints.

All computational tests presented in this thesis were performed on an Intel MacBook Pro (2.33 GHz dual core processor, 2GB RAM), and they were based on the implementation of the presented techniques within the Nektar++ framework.

Chapter 2

The Spectral/*hp* Element method

In this chapter, we give a brief overview of the fundamental concepts of the spectral/*hp* element method in order to facilitate the discussion in the forthcoming chapters. For a complete reference on this subject, the reader is referred to the textbook by Karniadakis & Sherwin (Karniadakis & Sherwin 2005). But as a start, we first review some terminology in order to situate the spectral/*hp* element method within the field of the finite element methods.

The finite element method (FEM) Nowadays, the finite element method is one of the most popular numerical methods in the field of both solid and fluid mechanics. It is a discretisation technique used to solve (a set of) partial differential equations in its equivalent variational form. The classical approach of the finite element method is to partition the computational domain into a mesh of many small subdomains and to approximate the unknown solution by piecewise linear interpolation functions, each with local support. The FEM has been widely discussed in literature and for a complete review of the method, the reader is also directed to the seminal work of Zienkiewicz and Taylor (Zienkiewicz & Taylor 1989).

High-order finite element methods While in the classical finite element method the solution is expanded in a series of linear basis functions, high-order FEMs employ higher-order polynomials to approximate the solution. For the high-order FEM, the solution is locally expanded into a set of $P + 1$ linearly independent polynomials

which span the polynomial space of order P. Confusion may arise about the use of the term *order*. While the order, or *degree*, of the expansion basis corresponds to the maximal polynomial degree of the basis functions, the order of the method essentially refers to the accuracy of the approximation. More specifically, it depends on the convergence rate of the approximation with respect to mesh-refinement. It has been shown in (Babuška & Suri 1994), that for a sufficiently smooth exact solution $u \in H^k(\Omega)$ $(k \geq P+1)$, the error of the FEM approximation u^δ can be bounded by

$$||u - u^\delta||_E \leq Ch^P ||u||_k. \tag{2.1}$$

This implies that when decreasing the mesh-size h, the error of the approximation algebraically scales with the P^{th} power of h. This can be formulated as

$$||u - u^\delta||_E = O(h^P). \tag{2.2}$$

If this holds, one generally classifies the method as a P^{th}-order FEM. However, for non-smooth problems, i.e. $k < P+1$, the order of the approximation will in general be lower than P, the order of the expansion.

h-version FEM A finite element method is said to be of h-type when the degree P of the piecewise polynomial basis functions is fixed and when any change of discretisation to enhance accuracy is done by means of a mesh refinement, that is, a reduction in h. Dependent on the problem, local refinement rather than global refinement may be desired. Consistent with the error estimate Eq. (2.1), the h-version of the classical FEM employing linear basis functions can be classified as a first-order method when resolving smooth solutions.

p-version FEM In contrast with the h-version FEM, finite element methods are said to be of p-type when the partitioning of domain is kept fixed and any change of discretisation is introduced through a modification in polynomial degree P. Again here, the polynomial degree may vary per element, particularly when the complexity of the problem requires local enrichment. However, sometimes the term p-type FEM is merely used to indicated that a polynomial degree of $P > 1$ is used.

***hp*-version FEM** In the *hp*-version of the FEM, both the ideas of mesh refinement and degree enhancement are combined.

The spectral method As opposed to the finite element methods which builds a solution from a sequence of local elemental approximations, spectral methods approximate the solution by a truncated series of *global* basis functions. Modern spectral methods, first presented in (Gottlieb & Orszag 1977), involve the expansion of the solution into high-order orthogonal expansions, typically by employing Fourier, Chebyshev or Legendre series.

The spectral element method Patera (Patera 1984) combined the high accuracy of the spectral methods with the geometric flexibility of the finite element method to form the spectral element method. The multi-elemental nature makes the spectral element method conceptually similar to the above mentioned high-order finite element. However, historically the term spectral element method has been used to refer to the high-order finite element method using a specific nodal expansion basis. The class of nodal higher-order finite elements which have become known as spectral elements, use the Lagrange polynomials through the zeros of the Gauss-Lobatto(-Legendre) polynomials.

The spectral/*hp* element method The spectral/*hp* element method, as its name suggests, incorporates both the multi-domain spectral methods as well as the more general high-order finite element methods. One can say that it encompasses all methods mentioned above. However, note that the term spectral/*hp* element method is mainly used in the field of fluid dynamics, while the terminology *p* and *hp*-FEM originates from the area of structural mechanics.

2.1 The Galerkin formulation

Finite element methods typically use the Galerkin formulation to derive the weak form of the partial differential equation to be solved. In this thesis, we will primarily

adopt the classical Galerkin formulation in combination with globally C^0 continuous spectral/*hp* element discretisations.

To describe the Galerkin method, consider a steady *linear* differential equation in a domain Ω denoted by

$$\mathbb{L}(u) = f, \tag{2.3}$$

subject to appropriate boundary conditions. In the Galerkin method, the weak form of this equation can be derived by pre-multiplying this equation with a test function v and integrating the result over the entire domain Ω to arrive at: Find $u \in \mathcal{U}$ such that

$$\int_\Omega v\mathbb{L}(u)d\boldsymbol{x} = \int_\Omega vfd\boldsymbol{x}, \quad \forall v \in \mathcal{V}, \tag{2.4}$$

where $\mathcal{U}$ and $\mathcal{V}$ respectively are a suitably chosen trial and test space (in the traditional Galerkin method, one typically takes $\mathcal{U} = \mathcal{V}$). In case the inner product of v and $\mathbb{L}(u)$ can be rewritten into a bi-linear form $a(v, u)$, this problem is often formulated more concisely as: Find $u \in \mathcal{U}$ such that

$$a(v, u) = (v, f), \quad \forall v \in \mathcal{V}, \tag{2.5}$$

where (v, f) denotes the inner product of v and f. The next step in the classical Galerkin finite element method is the discretisation: rather than looking for the solution u in the infinite dimensional function space $\mathcal{U}$, one is going to look for an approximate solution u^δ in the reduced finite-dimensional function space $\mathcal{U}^\delta \subset \mathcal{U}$. Therefore we represent the approximate solution as a linear combination of basis functions Φ_n that span the space $\mathcal{U}^\delta$, i.e.

$$u^\delta = \sum_{n\in\mathcal{N}} \Phi_n \hat{u}_n. \tag{2.6}$$

Adopting a similar discretisation for the test functions v, the discrete problem to be solved is given as: Find $\hat{u}_n$ $(n \in \mathcal{N})$ such that

$$\sum_{n\in\mathcal{N}} a(\Phi_m, \Phi_n)\hat{u}_n = (\Phi_m, f), \quad \forall m \in \mathcal{N}. \tag{2.7}$$

It is customary to describe this set of equations in matrix form as

$$\boldsymbol{A}\hat{\boldsymbol{u}} = \hat{\boldsymbol{f}}. \tag{2.8}$$

where $\hat{\boldsymbol{u}}$ is the vector of coefficients $\hat{u}_n$, $\boldsymbol{A}$ is the system matrix with elements

$$\boldsymbol{A}[m][n] = a(\Phi_m, \Phi_n) = \int_\Omega \Phi_m \mathbb{L}(\Phi_n) d\boldsymbol{x}, \tag{2.9}$$

and the vector $\hat{\boldsymbol{f}}$ is given by

$$\hat{\boldsymbol{f}}[m] = (\Phi_m, f) = \int_\Omega \Phi_m f d\boldsymbol{x}. \tag{2.10}$$

2.2 Elemental spectral/*hp* element discretisations

Like any finite element discretisation, a spectral/*hp* element discretisation starts by decomposing the domain Ω into a tessellation of $|\mathcal{E}|$ quadrilateral and/or triangular elements such that

$$\Omega = \bigcup_{e \in \mathcal{E}} \Omega_e. \tag{2.11}$$

Each of these elements Ω_e in *physical space* can be considered as an image of a standard element Ω_{st} in *reference space*. For every element, there exists a one-to-one mapping relating the physical Cartesian coordinates (x_1, x_2) of the element Ω_e to the reference coordinate system (ξ_1, ξ_2), which is defined as

$$x_1 = \chi_1^e(\xi_1, \xi_2), \quad x_2 = \chi_2^e(\xi_1, \xi_2). \tag{2.12}$$

2.2.1 Elemental expansion bases

For the two-dimensional spectral/*hp* expansions, we will only restrict our attention to *tensorial* expansions, that is, expansions which can be constructed as a tensor product of one-dimensional basis functions. This tensorial nature will be a necessary prerequisite for the application of the sum-factorisation technique in Section 5.1.1. Furthermore, we only consider expansion bases that have a typical *boundary/interior* decomposition such that they can easily be assembled into a globally C^0 continuous spectral/*hp* expansion.

2.2.1.1 Quadrilateral tensorial expansion bases

The quadrilateral standard element Ω_{st} is defined as the bi-unit square $\mathcal{Q}^2 = \{(\xi_1, \xi_2) \in [-1, 1] \times [-1, 1]\}$. Its Cartesian structure can be exploited to locally

represent the solution as a *tensorial* spectral/*hp* expansion

$$u(\xi_1, \xi_2) = \sum_{n \in \mathcal{N}} \phi_n(\xi_1, \xi_2)\hat{u}_n = \sum_{p=0}^{P} \sum_{q=0}^{P} \psi_p(\xi_1)\psi_q(\xi_2)\hat{u}_{pq}, \tag{2.13}$$

where the set of two-dimensional basis functions ϕ_n is defined as a tensor product of the one-dimensional basis functions ψ_p in each of the coordinate directions, as depicted in Fig. 2.1. The expansion basis ψ_p spans the polynomial space of order P and, for a globally C^0 continuous expansion – see also Section 2.3 – typically consists of a set of either *modal* or *nodal* basis functions which can be decomposed into *boundary modes* and *interior modes*. Boundary modes are defined as all the modes which have non-zero support on the boundary where interior modes are zero on all boundaries. As mentioned before, the tensor product nature of the expansion will be the necessary prerequisite for the application of the sum-factorisation technique outlined in Section 5.1.1.

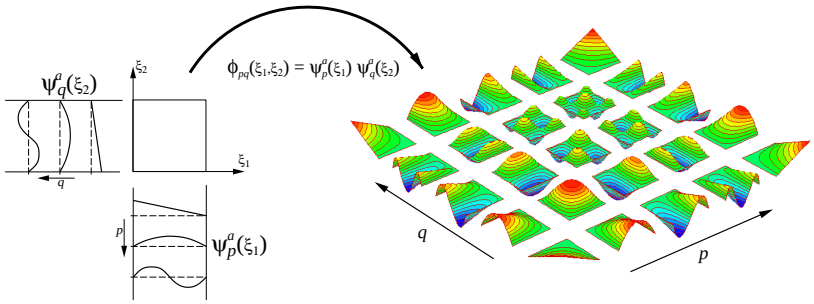


Figure 2.1: Construction of a two-dimensional C^0 continuous modal quadrilateral expansion basis from the tensor product of two one-dimensional expansions of order $P = 4$ (edge and face modes have been scaled by a factor of 4 and 16 respectively). Courtesy of (Karniadakis & Sherwin 2005).

2.2.1.2 Triangular tensorial expansion bases

The non-tensorial structure of the triangular reference element $\mathcal{T}^2 = \{-1 \leq \xi_1, \xi_2; \xi_1 + \xi_2 \leq 0\}$ seems to prohibit the construction of a tensorial expansion basis and consequently, the application of the sum-factorisation technique. However, after introducing a *collapsed* coordinate system given by the transformation

$$\eta_1(\xi_1, \xi_2) = 2\frac{1+\xi_1}{1-\xi_2} - 1, \qquad \eta_2(\xi_1, \xi_2) = \xi_2, \tag{2.14}$$

the triangular element can now be defined as $\mathcal{T}^2 = \{(\eta_1, \eta_2) \in [-1, 1] \times [-1, 1]\}$. In order to generate a C^0 continuous expansion and to ensure completeness of the expansion, we now use a *generalised* tensor product to define the expansion basis as

$$u(\xi_1, \xi_2) = \sum_{n \in \mathcal{N}} \phi_n(\xi_1, \xi_2) \hat{u}_n = \sum_{p=0}^{P} \sum_{q=0}^{f(p)} \psi_p \left(\eta_1(\xi_1, \xi_2)\right) \psi_{pq} \left(\eta_2(\xi_1, \xi_2)\right) \hat{u}_{pq}. \tag{2.15}$$

Note that the upper bound $f(p)$ of the index q now depends on the index p. Also the one-dimensional expansion basis $\{\psi_{pq}\}$ in the second coordinate direction ξ_2 now depends on both the indices p and q. This construction is graphically represented in Fig. 2.2 for the C^0 continuous modal triangular expansion introduced by Dubiner (Dubiner 1991). For an in-depth discussion of tensorial expansions for triangular elements, the reader is referred to (Karniadakis & Sherwin 2005).

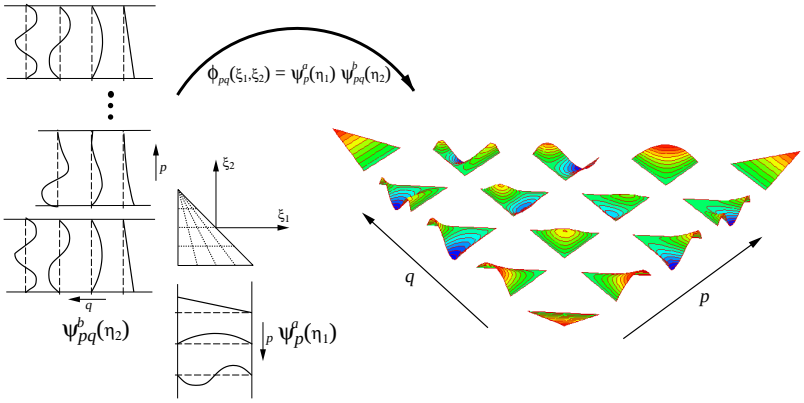


Figure 2.2: Construction of a two-dimensional 4^{th}-order C^0 continuous modal triangular expansion basis using a generalised tensor product procedure (edge and face modes have been scaled by a factor of 4 and 16 respectively). Courtesy of (Karniadakis & Sherwin 2005).

Note that in both the quadrilateral and triangular case, the lowest-order spectral/*hp* expansion (i.e. $P = 1$) corresponds to the well-known linear finite element expansion. The typical linear basis functions also appear as the vertex modes of higher-order modal spectral/*hp* expansions, as can be observed in Figs. 2.1 and 2.2.

2.2.2 Elemental operations within the standard region

The weak form of the problem in the finite element method typically requires the differentiation of functions and the evaluation of integrals over the domain. We

start by defining the discrete integration and differentiation operators within the standard region Ω_{st}. Furthermore, we also introduce the operator that we will refer to as the *backward transformation*.

2.2.2.1 Numerical integration

In order to allow for a numerical implementation of the FEM, different discrete integration rules, also known as *numerical quadrature*, have been proposed. In spectral/*hp* element methods, Gaussian quadrature is typically employed to evaluate integrals. The fundamental concept of Gaussian quadrature, which is particularly accurate when integrating smooth functions, is the approximation of the integral by a finite summation. In the one-dimensional standard region $[-1, 1]$, this yields the form (for Legendre integration),

$$\int_{-1}^{1} u(\xi) d\xi \approx \sum_{i=0}^{Q-1} \omega_i u(\xi_i), \tag{2.16}$$

where ω_i and ξ_i respectively are the weights and abscissas of the Q quadrature points within the standard segment. Dependent on the choice of the quadrature points, in particular the inclusion of the endpoints of the standard segment interval, one can distinguish between Gauss-Legendre, Gauss-Radau-Legendre and Gauss-Lobatto-Legendre quadrature. The use of Gaussian quadrature contributes to the efficiency of the FEM, as it permits the exact integration of polynomials of order exceeding $Q - 1$. Concretely, relation (2.16) is exact when,

- $u(\xi) \in \mathcal{P}_{2Q-1}([-1, 1])$ if using Gauss-Legendre quadrature,
- $u(\xi) \in \mathcal{P}_{2Q-2}([-1, 1])$ if using Gauss-Radau-Legendre quadrature,
- $u(\xi) \in \mathcal{P}_{2Q-3}([-1, 1])$ if using Gauss-Lobatto-Legendre quadrature.

This ensures that all discrete first-order and second-order linear operators in the spectral/*hp* element method will be evaluated exactly if the quadrature order is chosen to be at least:

- $Q = P + 1$ if using Gauss-Legendre or Gauss-Radau-Legendre quadrature,

- $Q = P + 2$ if using Gauss-Lobatto-Legendre quadrature.

However, note that the integration of, for example, quadratic non-linearities or the integration over curved elements may require a higher number of quadrature points (Kirby & Karniadakis 2003).

Numerical integration using Gaussian quadrature can be trivially extended to the two-dimensional standard regions, yielding

Quadrilateral region $\mathcal{Q}^2$

$$\int_{-1}^{1}\int_{-1}^{1} u(\xi_1,\xi_2)d\xi_1 d\xi_2 \approx \sum_{i=0}^{Q_1-1}\omega_i\left\{\sum_{j=0}^{Q_2-1}\omega_j u(\xi_{1_i},\xi_{2_j})\right\}, \tag{2.17}$$

Triangular region $\mathcal{T}^2$

$$\begin{aligned}\int_{\mathcal{T}^2} u(\xi_1,\xi_2)d\xi_1 d\xi_2 &= \int_{-1}^{1}\int_{-1}^{1} u(\eta_1,\eta_2)\left|\frac{\partial(\xi_1,\xi_2)}{\partial(\eta_1,\eta_2)}\right| d\eta_1 d\eta_2 \\ &\approx \sum_{i=0}^{Q_1-1}\omega_i\left\{\sum_{j=0}^{Q_2-1}\omega_j\frac{1-\eta_{2_j}}{2}u(\eta_{1_i},\eta_{2_j})\right\}, \\ &\approx \sum_{i=0}^{Q_1-1}\omega_i\left\{\sum_{j=0}^{Q_2-1}\bar{\omega}_j u(\eta_{1_i},\eta_{2_j})\right\},\end{aligned} \tag{2.18}$$

where $\bar{\omega}_j = \omega_j\frac{1-\eta_{2_j}}{2}$.

Gaussian quadrature: a remarkable property Gauss first recognised that for a particular choice of Q abscissas, it is possible to exactly integrate polynomials of order higher than $Q-1$. For Gauss-Legendre quadrature using Q quadrature points, he showed that any polynomial up to order $2Q-1$ can be integrated exactly.

However, when inverting this idea, the following remarkable fact can be recognised: every polynomial in the space $\mathcal{P}_{2Q-1}$ which has the same function values at the Q quadrature points, yields the same integral in the interval $[-1,1]$. This is demonstrated in Fig. 2.3 for second-order Gauss-Legendre integration. The points a and b, located at the two quadrature zeros $\xi = \pm\sqrt{\frac{1}{3}}$, uniquely define a linear polynomial and define an infinite set of quadratic and cubic polynomials. It is a

direct result from the properties of Gaussian quadrature, that all these polynomials (up to third-order) have the same integral in the interval $[-1, 1]$. This implies that the area of the shaded region in Fig. 2.3 is equal for every subfigure.

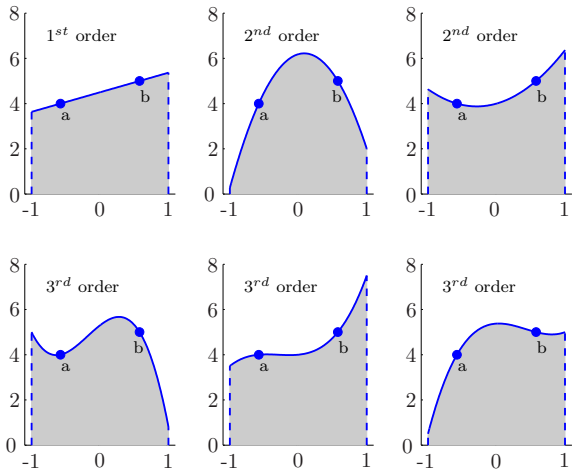


Figure 2.3: 2^{nd}-order Gauss-Legendre quadrature: All polynomials $u(\xi) \in \mathcal{P}_3([-1, 1])$ going through the points a and b yield the same value for $\int_{-1}^{1} u(\xi)d\xi$.

2.2.2.2 Collocation differentiation

Since a finite or spectral/*hp* element problem often involves the derivative of a function inside an integral, it can be appreciated that we require the value of the function derivative at the quadrature points in order to numerically evaluate the integral under consideration. We will employ a technique referred to as collocation differentiation in order to calculate these values.

Assume an arbitrary function $u(\xi)$, defined in the one-dimensional standard region $[-1, 1]$. This function can be approximated by expanding it as

$$u(\xi) \approx u^{\delta}(\xi) = \sum_{j=0}^{Q-1} h_j(\xi) u(\xi_j), \tag{2.19}$$

where $h_j(\xi)$ are the Lagrange polynomials through the set of Q quadrature (or *nodal*) points ξ_j, defined as

$$h_j(\xi) = \frac{\prod_{i=0, i\neq j}^{Q-1} (\xi - \xi_i)}{\prod_{i=0, i\neq j}^{Q-1} (\xi_j - \xi_i)}. \tag{2.20}$$

Note that this approximation is exact if $u(\xi)$ is a polynomial of order equal or less than $P = Q - 1$. The derivative of $u(\xi)$ at the quadrature points can then be evaluated as

$$\frac{\mathrm{d}u(\xi_i)}{\mathrm{d}\xi} \approx \frac{\mathrm{d}u^{\delta}(\xi_i)}{\mathrm{d}\xi} = \sum_{j=0}^{Q-1} d_{ij} u(\xi_j), \tag{2.21}$$

where the elements d_{ij} of the differentiation matrix are defined as

$$d_{ij} = \frac{\mathrm{d}h_j(\xi_i)}{\mathrm{d}\xi}. \tag{2.22}$$

This implies that the collocation differentiation technique allows us to compute the derivative of a function at the quadrature points based upon the function values based at the same points.

Quadrilateral region $\mathcal{Q}^2$ A two-dimensional function $u(\xi_1, \xi_2)$ defined on the unit-square $\mathcal{Q}^2$ can be represented in terms of Lagrange polynomials associated with the set of quadrature points as

$$u(\xi_1, \xi_2) \approx u^{\delta}(\xi_1, \xi_2) = \sum_{i=0}^{Q-1} \sum_{j=0}^{Q-1} h_i(\xi_1) h_j(\xi_2) u(\xi_{1_i}, \xi_{2_j}). \tag{2.23}$$

The partial derivative with respect to ξ_1 at the quadrature points can therefore be evaluated as

$$\frac{\partial u(\xi_{1_r}, \xi_{2_s})}{\partial \xi_1} \approx \frac{\partial u^{\delta}(\xi_{1_r}, \xi_{2_s})}{\partial \xi_1} = \sum_{i=0}^{Q-1} \sum_{j=0}^{Q-1} \frac{\mathrm{d}h_i(\xi_{1_r})}{\mathrm{d}\xi_1} h_j(\xi_{2_s}) u(\xi_{1_i}, \xi_{2_j}). \tag{2.24}$$

Due to the collocation property of the Lagrangian representation (that is, $h_j(\xi_s) = \delta_{js}$), this can be simplified to

$$\frac{\partial u^{\delta}(\xi_{1_r}, \xi_{2_s})}{\partial \xi_1} = \sum_{i=0}^{Q-1} \sum_{j=0}^{Q-1} d_{ri} \delta_{js} u(\xi_{1_i}, \xi_{2_j}) = \sum_{i=0}^{Q-1} d_{ri} u(\xi_{1_i}, \xi_{2_s}), \tag{2.25}$$

where d_{ri} is defined as in Eq. (2.22). The partial derivative with respect to ξ_2 can be calculated equivalently to arrive at

$$\frac{\partial u^{\delta}(\xi_{1_r}, \xi_{2_s})}{\partial \xi_2} = \sum_{i=0}^{Q-1} d_{si} u(\xi_{1_r}, \xi_{2_i}). \tag{2.26}$$

.

Triangular region $\mathcal{T}^2$ For the triangular standard region $\mathcal{T}^2$, we can represent any function $u(\xi_1, \xi_2)$ by a set of Lagrange polynomials in terms of the collapsed coordinates (η_1, η_2) as

$$u(\xi_1, \xi_2) \approx u^\delta(\xi_1, \xi_2) = \sum_{i=0}^{Q-1} \sum_{j=0}^{Q-1} h_i(\eta_1) h_j(\eta_2) u(\eta_{1_i}, \eta_{2_j}). \tag{2.27}$$

The partial derivatives with respect to the Cartesian system (ξ_1, ξ_2) may then be determined by applying the chain rule

$$\begin{bmatrix} \dfrac{\partial u^\delta}{\partial \xi_1} \\ \dfrac{\partial u^\delta}{\partial \xi_2} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \eta_1}{\partial \xi_1} & \dfrac{\partial \eta_2}{\partial \xi_1} \\ \dfrac{\partial \eta_1}{\partial \xi_2} & \dfrac{\partial \eta_2}{\partial \xi_2} \end{bmatrix} \begin{bmatrix} \dfrac{\partial u^\delta}{\partial \eta_1} \\ \dfrac{\partial u^\delta}{\partial \eta_2} \end{bmatrix} = \begin{bmatrix} \dfrac{2}{1-\eta_2} & 0 \\ \dfrac{1+\eta_1}{1-\eta_1} & 1 \end{bmatrix} \begin{bmatrix} \dfrac{\partial u^\delta}{\partial \eta_1} \\ \dfrac{\partial u^\delta}{\partial \eta_2} \end{bmatrix}, \tag{2.28}$$

where the value of the partial derivatives with respect to η_1 and η_2 at the quadrature points can be calculated as

$$\frac{\partial u^\delta(\eta_{1_r}, \eta_{2_s})}{\partial \eta_1} = \sum_{i=0}^{Q-1} d_{ri} u(\eta_{1_i}, \eta_{2_s}), \tag{2.29}$$

$$\frac{\partial u^\delta(\eta_{1_r}, \eta_{2_s})}{\partial \eta_2} = \sum_{i=0}^{Q-1} d_{si} u(\eta_{1_r}, \eta_{2_i}). \tag{2.30}$$

2.2.2.3 Backward transformation

When using modal expansion bases, it is often necessary to transform the coefficients of an expansion to the value of the spectral/*hp* expansion at the quadrature points. This is for example the case when applying the collocation differentiation technique to a spectral/*hp* expansion. This *backward transformation* from coefficient space to physical space is simply defined as

$$u(\xi_{1_i}, \xi_{2_j}) = \sum_{n \in \mathcal{N}} \phi_n(\xi_{1_i}, \xi_{2_j}) \hat{u}_n, \tag{2.31}$$

that is, the backward transformation is merely the evaluation of the spectral/*hp* element expansion at the quadrature points.

2.2.3 Elemental operations within general-shaped elements

We have mentioned earlier that every general-shaped element Ω_e can be seen as an image of the standard element Ω_{st} through the mapping of Eq. (2.12). As a result,

a spectral/*hp* expansion on a *local* element can be defined as

$$u(x_1, x_2) = \sum_{n \in \mathcal{N}} \phi_n \left(\chi_1^{-1}(x_1, x_2), \chi_2^{-1}(x_1, x_2) \right) \hat{u}_n. \tag{2.32}$$

where χ_1^{-1} and χ_2^{-1} together define the coordinate transformation from (x_1, x_2) to (ξ_1, ξ_2). In the spectral/*hp* element method, it is customary to express the mapping between the local Cartesian coordinates (x_1, x_2) and the reference coordinates (ξ_1, ξ_2) as a spectral/*hp* expansion of the same order as the expansion used to represent the solution, i.e.

$$x_i = \chi_i(\xi_1, \xi_2) = \sum_{n \in \mathcal{N}} \phi_n(\xi_1, \xi_2) \hat{x}_n^i. \tag{2.33}$$

When employing the modal hierarchical C^0 continuous basis functions for this *iso-parametric* mapping, the coefficients of all but the linear vertex modes can be zero in order to exactly describe a straight-sided element. To describe a curved region, however, requires more information such that we can truly speak from a high-order representation in that case.

2.2.3.1 Integration within a general-shaped element

In order to integrate a function $u(x_1, x_2)$ over an arbitrary quadrilateral or triangular region Ω_e, we can apply a coordinate transformation to arrive at

$$\int_{\Omega_e} u(x_1, x_2) dx_1 dx_2 = \int_{\Omega_{st}} u(\xi_1, \xi_2) \left| J \right| d\xi_1 d\xi_2, \tag{2.34}$$

where J is the Jacobian due to the transformation, defined as

$$J = \begin{vmatrix} \dfrac{\partial x_1}{\partial \xi_1} & \dfrac{\partial x_2}{\partial \xi_1} \\ \dfrac{\partial x_1}{\partial \xi_2} & \dfrac{\partial x_2}{\partial \xi_2} \end{vmatrix} = \frac{\partial x_1}{\partial \xi_1} \frac{\partial x_2}{\partial \xi_2} - \frac{\partial x_2}{\partial \xi_1} \frac{\partial x_1}{\partial \xi_2}. \tag{2.35}$$

The transformed integral is now over the reference region Ω_{st} such that we can apply the numerical quadrature rules of Section 2.2.2.1. Furthermore, the use of an iso-parametric mapping for the coordinate transformation allows us to calculate the terms $\frac{\partial x_i}{\partial \xi_j}$ in the Jacobian by means of the collocation differentiation technique of Section 2.2.2.2.

2.2.3.2 Differentiation within a general-shaped element

To differentiate a function $u(x_1, x_2)$ within an arbitrary region Ω_e, we apply the chain rule which, for the two-dimensional case, gives

$$\begin{bmatrix} \dfrac{\partial u}{\partial x_1} \\ \dfrac{\partial u}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \xi_1}{\partial x_1} & \dfrac{\partial \xi_2}{\partial x_1} \\ \dfrac{\partial \xi_1}{\partial x_2} & \dfrac{\partial \xi_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} \dfrac{\partial u}{\partial \xi_1} \\ \dfrac{\partial u}{\partial \xi_2} \end{bmatrix} = \frac{1}{J} \begin{bmatrix} \dfrac{\partial x_2}{\partial \xi_2} & -\dfrac{\partial x_2}{\partial \xi_1} \\ -\dfrac{\partial x_1}{\partial \xi_2} & \dfrac{\partial x_1}{\partial \xi_1} \end{bmatrix} \begin{bmatrix} \dfrac{\partial u}{\partial \xi_1} \\ \dfrac{\partial u}{\partial \xi_2} \end{bmatrix}. \tag{2.36}$$

Again we can use the collocation differentiation technique within a standard element to compute the metric terms $\frac{\partial \xi_i}{\partial x_j}$ as well as the derivatives of u with respect to the reference coordinates ξ_1 and ξ_2.

2.2.4 Elemental matrix and vector notation

The formulation in two dimensions and the use of tensorial expansions necessitates the introduction of a large number of subscript and superscript notations. To help clarify the operations to be discussed in the remainder of this work we, therefore, introduce a matrix and vector notation to represent operations such as integration, differentiation and the transformation from coefficient to physical space. We note, however, that the despite this matrix representation, the explicit construction of these matrices is not always necessary or even desirable in practice (see also Chapter 5). Finally, we also would like to note that all definitions in this section refer to a single element. In Section 2.3 we will extend the notation to include multiple elements. To distinguish between the two uses we will introduce the superscript e.

2.2.4.1 Elemental vectors

The vector of physical values $\boldsymbol{u}$ We define the vector $\boldsymbol{u}$ as the evaluation of a function $u(\boldsymbol{\xi})$ at the entire set of quadrature points, i.e.

$$\boldsymbol{u}[n] = u(\boldsymbol{\xi}_n). \tag{2.37}$$

When employing a tensor-product quadrature rule in two-dimensions, the quadrature points are ordered according to a lexicographical ordering along the ξ_1 direction

such that

$$\boldsymbol{u}[i+jQ_1] = u(\xi_{1_i}, \xi_{2_j}), \quad 0 \leq i < Q_1, 0 \leq j < Q_2. \tag{2.38}$$

This ordering is chosen to facilitate the application of the sum-factorisation technique as explained in Section 5.1.1.

The vector of coefficients $\hat{\boldsymbol{u}}$ To represent all expansion coefficients of a spectral/*hp* element discretisation in vector form, we adopt a similar notation but with a circumflex, such that

$$\hat{\boldsymbol{u}}[n] = \hat{u}_n, \quad n \in \mathcal{N}. \tag{2.39}$$

For the tensor-based quadrilateral expansion coefficients $\hat{u}_{pq}$ we again adopt a lexicographical numbering convention along the ξ_1 direction, yielding

$$\hat{\boldsymbol{u}}[p+q(P_1+1)] = \hat{u}_{pq}, \quad 0 \leq p \leq P_1, 0 \leq q \leq P_2. \tag{2.40}$$

For triangular tensorial expansions, however, the application of the sum-factorisation technique requires a lexicographical ordering along the ξ_2 direction, i.e.

$$\hat{\boldsymbol{u}}[q + \frac{p(2P_2+3-p)}{2}] = \hat{u}_{pq}, \quad 0 \leq p, q,\, p \leq P_1,\, p+q \leq P_2,\, P_1 \leq P_2. \tag{2.41}$$

2.2.4.2 Elemental matrices

The basis matrix $\boldsymbol{B}$ The basis matrix $\boldsymbol{B}$ can be considered as a discrete representation of the basis functions and is therefore defined as having columns which are fixed expansion modes $\phi_n(\boldsymbol{\xi})$ evaluated at all the quadrature points $\boldsymbol{\xi}_m$, that is,

$$\boldsymbol{B}[m][n] = \phi_n(\boldsymbol{\xi}_m). \tag{2.42}$$

Note that both the modes (the columns) and quadrature points (the rows) within the matrix $\boldsymbol{B}$ are ordered in a consistent fashion to the vectors $\hat{\boldsymbol{u}}$ and $\boldsymbol{u}$ respectively.

The weight matrix $\boldsymbol{W}$ The weight matrix $\boldsymbol{W}$ is a diagonal matrix containing the Gaussian quadrature weights multiplied by the Jacobian at the quadrature points and is ordered consistent with the vector $\boldsymbol{u}$, such that

$$\boldsymbol{W}[m][m] = \omega_m \left| J(\boldsymbol{\xi}_m) \right|, \tag{2.43}$$

where in two dimensions, ω_m is the product of the one-dimensional quadrature weights in each of the coordinate directions evaluated at $\boldsymbol{\xi}_m$.

The differentiation matrices $\boldsymbol{D}$ The differentiation matrices $\boldsymbol{D}_{\xi_1}$ and $\boldsymbol{D}_{\xi_2}$ are defined as

$$\boldsymbol{D}_{\xi_1} = \boldsymbol{I}_{Q_1} \otimes \boldsymbol{D}^{1d}_{\xi_1}, \tag{2.44}$$

$$\boldsymbol{D}_{\xi_2} = \boldsymbol{D}^{1d}_{\xi_2} \otimes \boldsymbol{I}_{Q_2}, \tag{2.45}$$

where $\boldsymbol{I}_Q$ is the identity matrix of size $Q \times Q$, $\otimes$ denotes the Kronecker product and $\boldsymbol{D}^{1d}_{\xi_i}$ is the one-dimensional differentiation matrix (see also Eq. (2.22)) with entries

$$\boldsymbol{D}^{1d}_{\xi_i}[m][n] = \frac{\mathrm{d}h_n(\xi_{i_m})}{\mathrm{d}\xi_i}. \tag{2.46}$$

Furthermore, we define the differentiation matrices $\boldsymbol{D}_{x_1}$ and $\boldsymbol{D}_{x_2}$ as

$$\boldsymbol{D}_{x_1} = \boldsymbol{\Xi}^1_1 \boldsymbol{D}_{\xi_1} + \boldsymbol{\Xi}^2_1 \boldsymbol{D}_{\xi_2}, \tag{2.47a}$$

$$\boldsymbol{D}_{x_2} = \boldsymbol{\Xi}^1_2 \boldsymbol{D}_{\xi_1} + \boldsymbol{\Xi}^2_2 \boldsymbol{D}_{\xi_2}, \tag{2.47b}$$

where $\boldsymbol{\Xi}^i_j$ are the diagonal matrices containing the derivative metrics evaluated at the quadrature points, i.e.

$$\boldsymbol{\Xi}^i_j[m][m] = \left.\frac{\partial \xi_i}{\partial x_j}\right|_{\boldsymbol{\xi}_m}. \tag{2.48}$$

2.2.4.3 Elemental operations in matrix notation

The introduction of a vector and matrix notation allows to formulate the earlier defined operations such as the backward transformation, integration and differentiation in a more concise way.

Backward transformation The backward transformation from coefficient to physical space in matrix notation can be represented as

$$\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}. \tag{2.49}$$

Integration - Inner product In a finite element method, the integrals to be evaluated are typically of the form, see Eq. (2.10),

$$\hat{\boldsymbol{I}}[n] := \int_{\Omega_e} \phi_n(x_1, x_2) u(x_1, x_2) dx_1 dx_2, \quad \forall n \in \mathcal{N}, \tag{2.50}$$

which can be recognised as the inner product of a function u with respect to all the basis functions ϕ_n. The discrete evaluation of this operation can be represented in matrix notation as

$$\hat{\boldsymbol{I}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{u}. \tag{2.51}$$

Differentiation Following the collocation differentiation technique, the derivative of a function u evaluated at the quadrature nodes can be formulated in matrix notation as

$$\frac{\partial \boldsymbol{u}}{\partial \xi_i} = \boldsymbol{D}_{\xi_i} \boldsymbol{u}, \tag{2.52}$$

$$\frac{\partial \boldsymbol{u}}{\partial x_i} = \boldsymbol{D}_{x_i} \boldsymbol{u} = \sum_{j=1}^{2} \boldsymbol{\Xi}_i^j \boldsymbol{D}_{\xi_j} \boldsymbol{u}. \tag{2.53}$$

2.3 Global spectral/*hp* element discretisations

2.3.1 Global expansion functions

Although high-order expansion bases are initially constructed on an elemental level, we require some form of connectivity between the elements in order to solve partial differential equations. Such global representations of the solution are typically constructed by imposing C^0 continuity across element boundaries. This procedure is facilitated by the boundary/interior decomposition of the elemental modes as we consequently only need to merge the corresponding boundary modes of adjacent elements into single global modes. This is depicted in Fig. 2.4(a). A global C^0 continuous spectral/*hp* expansion can then be represented as

$$u(x_1, x_2) = \sum_{m \in \mathcal{N}^g} \Phi_m(x_1, x_2) \hat{u}_m^g = \sum_{e \in \mathcal{E}} \sum_{n \in \mathcal{N}} \phi_n^e(x_1, x_2) \hat{u}_n^e, \tag{2.54}$$

where $\hat{u}_i^g$ are the global degrees of freedom corresponding to the global expansion basis Φ_i. The $|\mathcal{E}| \times |\mathcal{N}|$ elemental degrees of freedom $\hat{u}_m^e$ can be related to the $|\mathcal{N}^g|$

global degrees of freedom $\hat{u}_n^g$ through the local-to-global mapping $m(e,n)$. This many-to-one mapping can also be represented by the matrix operation $\boldsymbol{\mathcal{A}}$, that is,

$$\hat{\boldsymbol{u}}_l = \boldsymbol{\mathcal{A}}\hat{\boldsymbol{u}}_g, \tag{2.55}$$

which can be seen as scattering the vector of global coefficients $\hat{\boldsymbol{u}}_g$ upon the vector of local coefficients $\hat{\boldsymbol{u}}_l$. We can also write the vector $\hat{\boldsymbol{u}}_l$ as

$$\hat{\boldsymbol{u}}_l = \underline{\hat{\boldsymbol{u}}}^e = \begin{bmatrix} \hat{\boldsymbol{u}}^1 \\ \hat{\boldsymbol{u}}^2 \\ \vdots \\ \hat{\boldsymbol{u}}^{|\mathcal{E}|} \end{bmatrix}, \tag{2.56}$$

where we have adopted the notation that an underlined vector denotes an extension over all elemental regions.

2.3.2 Global operations

When following a traditional Galerkin procedure, the discrete weak formulation to be solved is an integral form which typically contains terms of the form

$$\hat{\boldsymbol{I}}_g[m] := \int_\Omega \Phi_m(x_1,x_2)u(x_1,x_2)dx_1dx_2, \qquad m \in \mathcal{N}^g. \tag{2.57}$$

The elemental decomposition of the problem now allows us to express this inner product of the function $u(x_1,x_2)$ with respect to the global basis functions Φ_m as a series of elemental contributions such that

$$\hat{\boldsymbol{I}}_g[m] := \sum \int_{\Omega^e} \phi_n^e(x_1,x_2)u(x_1,x_2)dx_1dx_2, \tag{2.58}$$

where the summation is taken over these elemental ϕ_n^e modes that correspond to the m^{th} global mode as defined by the mapping $m(e,n)$. This process, referred to as *global assembly* or *direct stiffness summation* is graphically represented in Fig. 2.4(b) and can be mathematically expressed as the transpose of $\boldsymbol{\mathcal{A}}$ such that

$$\hat{\boldsymbol{I}}_g = \boldsymbol{\mathcal{A}}^\top \hat{\boldsymbol{I}}_l, \tag{2.59}$$

where $\hat{\boldsymbol{I}}_l = \underline{\hat{\boldsymbol{I}}}^e$ is the vector of local contributions. This procedure of global assembly

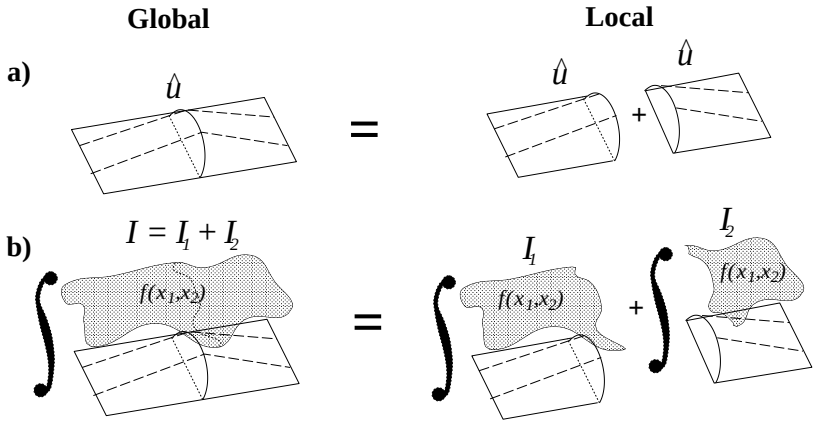


Figure 2.4: Illustration of local to global assembly. If we have a global expansion as represented in figure (a) it can be decomposed into two elemental contributions multiplied by the same global coefficient $\hat{u}$. To integrate a function $f(x_1, x_2)$ with respect to the global mode, as illustrated in figure (b), the integration in the global region is the sum of the integration in the local regions.

can also be used to construct global system matrices out of elemental contributions. For example the global mass matrix, defined as

$$\boldsymbol{M}[m][n] = \int_{\Omega} \Phi_m(x_1, x_2)\Phi_m(x_1, x_2)dx_1dx_2, \tag{2.60}$$

can be constructed as

$$\boldsymbol{M} = \boldsymbol{\mathcal{A}}^{\top}\underline{\boldsymbol{M}}^{e}\boldsymbol{\mathcal{A}}, \tag{2.61}$$

where we have adopted the notation that an underlined matrix denotes a block-diagonal concatenation of elemental matrices, i.e.

$$\underline{\boldsymbol{M}}^{e} = \begin{bmatrix} \boldsymbol{M}^1 & 0 & 0 & \cdots & 0 \\ 0 & \boldsymbol{M}^2 & 0 & \cdots & 0 \\ 0 & 0 & \boldsymbol{M}^3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \boldsymbol{M}^{|\mathcal{E}|} \end{bmatrix}. \tag{2.62}$$

Chapter 3

Encapsulation of Spectral/*hp* Elements

A major challenge which arises when one aims to develop a software package that implements the spectral/*hp* element method is to implement the mathematical structure of the method in a digestible and coherent matter. Obviously, there are many ways to encapsulate the fundamental concepts related to the spectral/*hp* element method, depending on e.g. the intended goal of the developer or the chosen programming language. In this chapter, we will –without going in too much detail– give a an overview of how we have chosen to abstract and implement spectral/*hp* elements in the *Nektar++* library (Kirby & Sherwin 2006). However, we want to emphasise that this is not the only possible choice.

3.1 Nektar++ and the abstraction of spectral/*hp* elements

Nektar++ (Kirby & Sherwin 2006) is an open source software library currently being developed at Imperial College London in collaboration with the University of Utah. It is designed to provide a toolbox of data structures and algorithms which implement the spectral/*hp* element method. Nektar++ is the continuation and adaptation of the Nektar flow solver. As opposed to its predecessor which focused

on solving fluid dynamics problems, Nektar++ is implemented as a C++ object-oriented toolkit which allows developers to implement spectral element solvers for a variety of different engineering problems. Nektar++ heavily relies on the object-oriented nature of the C++ programming language. It is designed as a collection of classes that extensively make use of powerful features such as inheritance, polymorphism, virtual functions, templates, ... For more information about these topics, the reader is referred to (Stroustrup 2000).

The structure of the Nektar++ library, a collection of five different sublibraries, is designed to reflect the typical structure of a global spectral/*hp* approximation. Two of the main characteristics of this typical structure are:

- **The elemental decomposition of the problem**

  As for all finite element methods, the computational domain is partitioned into a mesh of many small subdomains or elements. Analogously, the spectral/*hp* solution is expanded into a series of local expansions, each with support on a single element. This elemental representation enables the treatment of operations on a local elemental basis rather than on global level. This not only simplifies the formulation but also allows many operations to be performed more efficiently.

- **The introduction of a standard region**

  The introduction of a standard region allows the expansion basis to be defined just once, that is only on the standard region. All other elements then can be considered as the image of the standard element under a parametric mapping. Consequently, the elemental operations of integration and differentiation can all be executed on the standard element, subject to a proper treatment of the transformation from local (world space) to standard (reference space) coordinates. For curved-sided elements, the mapping from standard element to local element is generally done using an *iso-parametric* representation. In this case, the local geometry is represented with an expansion of the same form and polynomial order as the unknown variables.

This structure, supplemented with building blocks such as block matrix linear alge-

bra routines and automatic data coordinating objects, allows for an encapsulation in an object-oriented C++ implementation. Five different sublibraries, employing this characteristic pattern, are provided in the full Nektar++ library:

- the standard elemental region sublibrary (*StdRegions library*)
- the parametric mapping sublibrary (*SpatialDomains library*)
- the local elemental region sublibrary (*LocalRegions library*)
- the global region sublibrary (*MultiRegions library*)
- the supporting utilities sublibrary (*LibUtilities library*)

This structure can also be related to the formulation of a global spectral/*hp* element expansion, i.e.

$$u(\boldsymbol{x}) = \overbrace{\sum_{e\in\mathcal{E}} \underbrace{\sum_{n\in\mathcal{N}} \phi_n^e(\boldsymbol{x})\hat{u}_n^e}_{\text{LocalRegions library}}}^{\text{MultiRegions library}} = \sum_{e\in\mathcal{E}} \underbrace{\sum_{n\in\mathcal{N}} \phi_n^{std} \overbrace{\left([\chi^e]^{-1}(\boldsymbol{x})\right)}^{\text{SpatialDomains library}} \hat{u}_n^e}_{\text{StdRegions library}}. \tag{3.1}$$

A more detailed overview of the Nektar++ structure, including an overview of the most important classes per sublibrary, is depicted in Fig. 3.1. The idea behind these classes will be briefly outlined in the sections below.

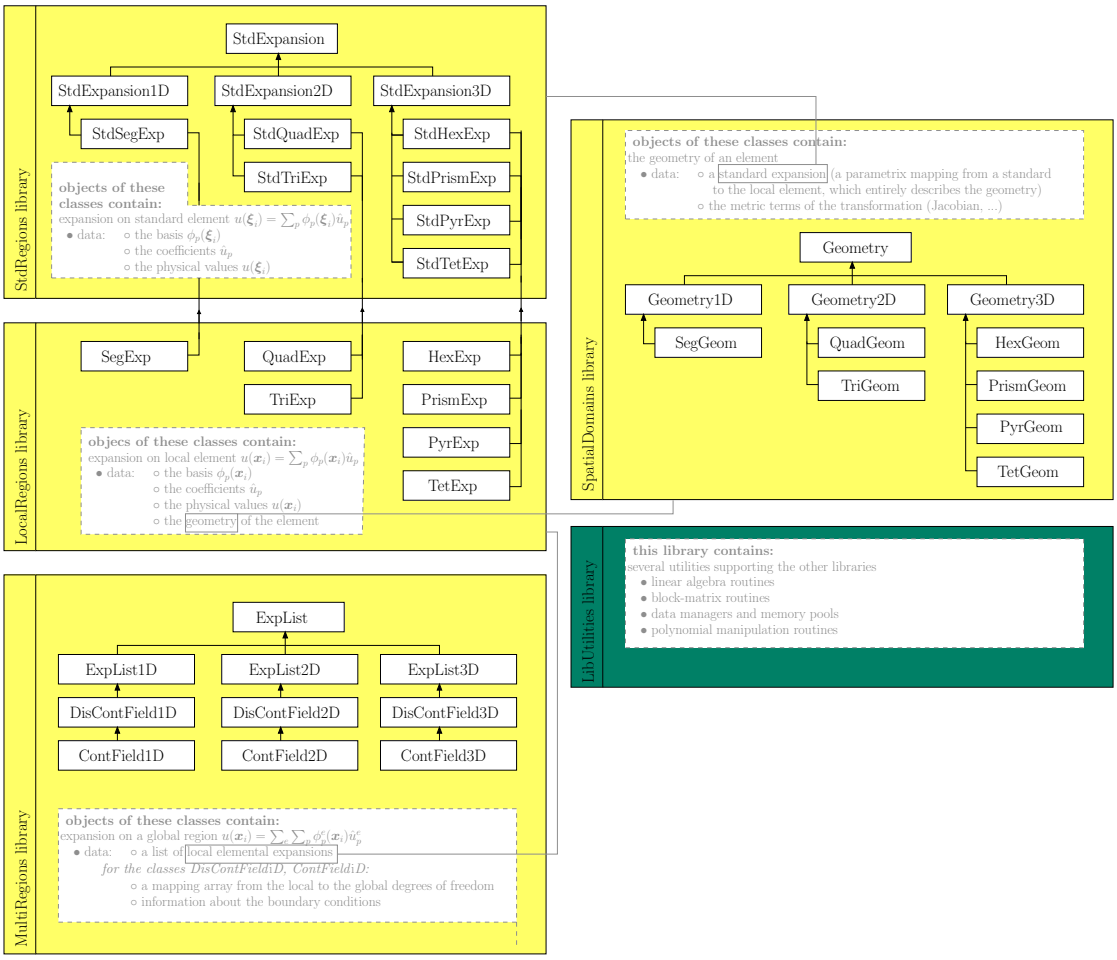


Figure 3.1: Main structure of the Nektar++ library.

3.1.1 The StdRegions library

The StdRegions library, see Fig. 3.1, bundles all classes that mimic a spectral/*hp* element expansion on a standard region. Such an expansion, i.e.

$$u(\boldsymbol{\xi}_i) = \sum_{n \in \mathcal{N}} \phi_n(\boldsymbol{\xi}_i)\hat{u}_n, \tag{3.2}$$

can be encapsulated in a class that essentially should only contain three data structures, respectively representing (see also Section 2.2.4):

- the coefficient vector $\hat{\boldsymbol{u}}$,
- the discrete basis matrix $\boldsymbol{B}$, and

- the vector $\boldsymbol{u}$ which represents the value of the expansion at the quadrature points $\boldsymbol{\xi}_i$.

All standard expansions, independent of the dimensionality or shape of the standard region, can be abstracted in a similar way. Therefore, it is possible to define these data structures in an *abstract* base class, i.e. the class `StdExpansion`. This base class can also contain the implementation of methods that are identical across all shapes. Derived from this base class is another level of abstraction, i.e. the abstract classes `StdExpansion1D`, `StdExpansion2D` and `StdExpansion3D`. All other shape-specific classes (such as e.g. `StdSegExp` or `StdQuadExp`) are inherited from these abstract base classes. These shape-specific classes are the classes from which objects will be instantiated. They also contain the shape-specific implementation for operations such as integration or differentiation.

3.1.2 The SpatialDomains library

The most important family of classes in the Spatialdomains library is the `Geometry` family, as can also be seen in Fig. 3.1. These classes are the representation of a (geometric) element in *physical space*. It has been indicated before, see Section 2.2.1, that every local element can be considered as an image of the standard element where the corresponding one-to-one mapping can be represented as an elemental standard spectral/*hp* expansion. As such, a proper encapsulation should at least contain data structures that represent such an expansion in order to completely define the geometry of the element. Therefore, we have equipped the classes in the `Geometry` family with the following data structures:

- an object of `StdExpansion` class, and
- a data structure that contains the metric terms (Jacobian, derivative metrics, ...) of the transformation.

Note that although the latter data structure is not necessary to define the geometry, it contains information inherent to the iso-parametric representation of the element that can later be used in e.g. the LocalRegions library. Again, the `StdExpansion`

object can be defined in the abstract base class `Geometry`. However, for every shape-specific geometry class, it needs to be initialised according to the corresponding StdRegions class (e.g. for the `QuadGeom` class, it needs to be initialised as an `StdQuadExp` object).

3.1.3 The LocalRegions library

The LocalRegions library is designed to encompass all classes that encapsulate the elemental spectral/*hp* expansions in physical space, see also Fig. 3.1. Considering Eq. (3.1), it can be appreciated that such a local expansion essentially *is a* standard expansion that *has a* (in C++ parlance) additional coordinate transformation that maps the standard element to the local element. In an object-oriented context, these *is-a* and *has-a* relationships can be applied as follows: the classes in the LocalRegions library are derived from the `StdExpansion` class tree but they are supplied with an additional data member representing the geometry of the local element. Depending on the shape-specific class in the LocalRegions library, this additional data member is an object of the corresponding class in the `Geometry` class structure. This inheritance between the LocalRegions and StdRegions library also allows for a localised implementation that prevents code duplication. In order to e.g. evaluate the integral over a local element, the integrand can be multiplied by the Jacobian of the coordinate transformation, where after the evaluation is redirected to the StdRegions implementation.

3.1.4 The MultiRegions library

The MultiRegions library (see Fig. 3.1) is the sublibrary that contains the abstraction of multi-elemental and global spectral/*hp* element expansions. In particular, it is the class `ExpList` and its derived classes that have been designed for this intended goal. Due to the modular structure of the previously introduced elemental sublibraries, this base class can be thought of as nothing more as an array of local expansions. As a results, the main data structure in the base class effectively is an array of `StdExpansion` objects (however, these objects will have to be initialised as

objects of the LocalRegions classes). Also the implementation of operations such as integration or differentiation within this class simply can be defined as a loop over the elemental implementations. The use of virtual functions in the StdRegions and LocalRegions will ensure that the appropriate shape-specific elemental implementation will be selected.

As a multi-elemental spectral/*hp* expansion often is more than just a collection of separate elements, we have also derived the `DisContField` and `ContField` classes (with the classes `ExpList`i`D` as a layer of abstraction in between). These classes additionally contain information about the connectivity between the elements which is encapsulated as a mapping from local to global degrees of freedom. In terms of connectivity, we can make a distinction between Discontinuous Galerkin expansions (`DisContField`) and the C^0 continuous Galerkin expansions (`ContField`). The `DisContField` and `ContField` classes are also the classes that are intended to be used by the user to solve partial differential equations.

3.1.5 Using the libraries

These libraries can be used as ingredients for the driving application, for example, a Navier-Stokes solver. This conceptual approach of the software leads to generic implementations with a high user-flexibility. A two-dimensional elliptic Helmholtz solver can for example be easily implemented as (in simplified pseudo C++ code)

```
ExpList* Exp = new ContField2D(inputfile);
Exp->HelmSolve();
```

However, one can easily change this into a 3D discontinuous Galerkin solver by simply replacing the first line by

```
ExpList* Exp = new DisContField3D(inputfile);
```

3.2 Good coding practice

One of the challenges in object-oriented programming is finding a good balance between a generic code architecture on one side and the performance on the other

side. While the former has been discussed in the previous sections, we here want to focus on some of the aspects that may help to minimise the run-time of your code. The goal of this section is not to investigate how to improve performance by means of different implementation strategies that are conceptually different (this will be the subject of the forthcoming chapters). We merely want to provide some practical guidelines that, once you have selected an algorithm to implement, may help you to write efficient code (in the sense that it runs fast).

These guidelines have resulted from our attempt to optimise the implementation of the elemental operators. Despite the fact that some of the guidelines may be obvious for experienced programmers, we do want to share them as we have experienced that they all together have lead to a significant speed-up. And although they are more generally valid, we will introduce the guidelines by means of the example of the elemental weak Laplacian operator, defined as

$$\hat{v}_m = \sum_{n \in \mathcal{N}} \left(\nabla \phi_m, \nabla \phi_n \right) \hat{u}_m, \quad \forall m \in \mathcal{N}. \tag{3.3}$$

where $(\cdot, \cdot)$ represents the inner product over the element Ω_e. The weak Laplacian operator can also be seen as part of the weak Helmholtz operator, an essential building block in the solution procedure of the advection-diffusion equation. Note that we will focus on the matrix free (using sum-factorisation, see also Chapter 5) evaluation of this operator.

3.2.1 Implement the most efficient formulation

For a two-dimensional quadrilateral element, Eq. (3.3), can be rewritten as

$$\hat{v}_m = \sum_{n \in \mathcal{N}} \left\{ \left(\frac{\partial \phi_m}{\partial x_1}, \frac{\partial \phi_n}{\partial x_1} \right) + \left(\frac{\partial \phi_m}{\partial x_2}, \frac{\partial \phi_n}{\partial x_2} \right) \right\} \hat{u}_m, \quad \forall m \in \mathcal{N}. \tag{3.4}$$

A first implementation, let us call it the naive implementation, can be based upon an equivalent formulation in terms of the matrix operator $\boldsymbol{D}_{x_i}$, $\boldsymbol{B}$ and $\boldsymbol{W}$, yielding

$$\hat{\boldsymbol{v}} = \left(\boldsymbol{D}_{x_1} \boldsymbol{B} \right)^\top \boldsymbol{W} \boldsymbol{D}_{x_1} \boldsymbol{B} \hat{\boldsymbol{u}} + \left(\boldsymbol{D}_{x_2} \boldsymbol{B} \right)^\top \boldsymbol{W} \boldsymbol{D}_{x_2} \boldsymbol{B} \hat{\boldsymbol{u}}, \tag{3.5}$$

where $\boldsymbol{D}_{x_i}$ is the discrete derivative operator with respect to the i^{th} coordinate direction and $\boldsymbol{B}$ and $\boldsymbol{W}$ are the operators as defined in Section 2.2.4. To demonstrate

our point of implementing the most efficient formulation, let us focus on the derivative operator. Starting from the expression above, it is possible to implement the Laplacian operator making four separate function calls to the local derivative operator, twice with respect to x_1 and twice with respect to x_2. However underneath, each derivative operator $\boldsymbol{D}_{x_i}$ will be evaluated in terms of the standard derivative operators $\boldsymbol{D}_{\xi_i}$, according to Eq. (2.47), as

$$\boldsymbol{D}_{x_i} = \sum_{j=1}^{2} \boldsymbol{\Xi}_i^j \boldsymbol{D}_{\xi_j}, \tag{3.6}$$

where we have previously in Section 2.2.4 defined $\boldsymbol{\Xi}_i^j$ as the diagonal matrix of the derivative metrics evaluated at the quadrature points, i.e. $\boldsymbol{\Xi}_i^j[k][k] = \left.\frac{\partial \xi_j}{\partial x_i}\right|_{\boldsymbol{\xi}_k}$. As a result, such a naive (but generic) implementation of the weak Laplacian operator will actually require $4 \times 2 = 8$ derivative operations in total.

The problem with the naive implementation is that it evaluates the standard derivative operators $\boldsymbol{D}_{\xi_i}$ more than strictly necessary. To appreciate this, insert Eq. (3.6) into Eq. (3.5), to arrive at

$$\hat{\boldsymbol{v}} = \sum_{i=1}^{2} \left[\left(\sum_{j=1}^{2} \boldsymbol{\Xi}_i^j \boldsymbol{D}_{\xi_j} \right) \boldsymbol{B} \right]^\top \boldsymbol{W} \left(\sum_{k=1}^{2} \boldsymbol{\Xi}_i^k \boldsymbol{D}_{\xi_j} \right) \boldsymbol{B}\hat{\boldsymbol{u}}. \tag{3.7}$$

After appropriate rearranging, this leads to the formulation (see also Appendix A.2.5)

$$\hat{\boldsymbol{v}} = \boldsymbol{B}^\top \begin{bmatrix} \boldsymbol{D}_{\xi_1}^\top & \boldsymbol{D}_{\xi_2}\top \end{bmatrix} \begin{bmatrix} \sum_{i=1}^{2} \boldsymbol{\Xi}_i^1 \boldsymbol{\Xi}_i^1 \boldsymbol{W} & \sum_{i=1}^{2} \boldsymbol{\Xi}_i^1 \boldsymbol{\Xi}_i^2 \boldsymbol{W} \\ \sum_{i=1}^{2} \boldsymbol{\Xi}_i^2 \boldsymbol{\Xi}_i^1 \boldsymbol{W} & \sum_{i=1}^{2} \boldsymbol{\Xi}_i^2 \boldsymbol{\Xi}_i^2 \boldsymbol{W} \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_{\xi_1} \\ \boldsymbol{D}_{\xi_2} \end{bmatrix} \boldsymbol{B}\hat{\boldsymbol{u}}. \tag{3.8}$$

It can be observed that the implementation of the weak Laplacian operator in essence only requires four standard derivative operations, rather than the required eight in the naive implementation. The optimal implementation should then best be based upon the formulation given by Eq. (3.8) rather than on the formulation as given by Eq. (3.5). It may be appreciated that for this example, selecting the most efficient formulation may lead to a significant performance gain.

3.2.2 Precompute and store relevant information

Considering Eq. (3.6), we see that the local derivative operator requires the derivative metrics $\boldsymbol{\Xi}_i^j$ that are associated to the mapping from the reference element to the local element. In general these metric terms are computed only once, that is when preprocessing the mesh. They subsequently are stored for future use. From Eq. (3.8), it can be seen that the matrices $\boldsymbol{\Xi}_i^j$ also come back in the evaluation of the weak Laplacian operator. However they do not appear separately but they are combined together with the quadrature metric $\boldsymbol{W}$ to form some kind of *Laplacian metric* $\boldsymbol{G}^{ij}$, defined as

$$\boldsymbol{G}^{ij} = \sum_{k=1}^{2} \boldsymbol{\Xi}_k^i \boldsymbol{\Xi}_k^j \boldsymbol{W}. \tag{3.9}$$

For an efficient evaluation of the weak Laplacian operator, it is advised to also precompute and store these Laplacian metrics $\boldsymbol{G}^{ij}$ during preprocessing. The weak Laplacian operator then can be formulated as

$$\hat{\boldsymbol{v}} = \boldsymbol{B}^\top \begin{bmatrix} \boldsymbol{D}_{\xi_1}^\top & \boldsymbol{D}_{\xi_2}\top \end{bmatrix} \begin{bmatrix} \boldsymbol{G}^{11} & \boldsymbol{G}^{12} \\ \boldsymbol{G}^{21} & \boldsymbol{G}^{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_{\xi_1} \\ \boldsymbol{D}_{\xi_2} \end{bmatrix} \boldsymbol{B}\hat{\boldsymbol{u}}. \tag{3.10}$$

Although the on-the-fly computation of the terms $\boldsymbol{G}^{ij}$ in essence merely is an $\mathcal{O}(P^2)$ operation (compared to the $\mathcal{O}(P^3)$ operators $\boldsymbol{B}$ and $\boldsymbol{D}_{\xi_i}$ – see also Section 5.2 and Appendix A for more information on operation count), it in fact is an rather expensive operation as it involves at least 12 separate $\mathcal{O}(P^2)$ operations (vector additions and/or vector multiplications) in total. Although the storage of these additional terms $\boldsymbol{G}^{ij}$ requires extra memory, it will significantly speed-up the execution.

3.2.3 Implement specialised vector algebra routines

Although we just explained that for optimal efficiency it is not advised to compute the terms $\boldsymbol{G}^{ij}$ on-the-fly, it provides an ideal example to illustrate the current guideline. Although we have introduced all the terms in the expression

$$\boldsymbol{G}^{ij} = \left(\boldsymbol{\Xi}_1^i \boldsymbol{\Xi}_1^j + \boldsymbol{\Xi}_2^i \boldsymbol{\Xi}_2^j\right) \boldsymbol{W}, \tag{3.11}$$

as diagonal matrices, they will of coarse be encapsulated as some sort of vector or array. As a result, the computation of $\boldsymbol{G}^{ij}$ is a combination of vector additions and

vector multiplications. In the Nektar++ library, we previously used the available BLAS *level I* (Dongarra, Du Croz, Hammarling, Hanson, & Duff 1988) type of routines, such as the `daxpy` routine or similar routines as provided by other vector manipulation routines. However, these routines mainly operate on a limited amount of input vectors (typically two or three). This means that in order to evaluate the entire right-hand-side of Eq. (3.11), it is required to make repetitive calls to these functions as well as to use some temporary memory storage to save the intermediate results. This can be optimised by extending these existing libraries with your own routines. For the example under consideration, we have observed that a tailor-made routine as simple as

```
for(int i = 0; i < size; i++)
{
    z[i] = (a[i]*b[i]+c[i]*d[i])*e[i];
}
```

is sufficient to seriously enhance the performance. Although the number of floating point operations is identical, the performance difference can mainly be attributed to the reduction in the number of memory references and the circumvention of temporary memory storage.

3.2.4 Smart use of temporary memory storage

Inefficient use and allocation of temporary memory storage may hamper the performance. This is especially the case for low-order expansions, in which the cost associated to temporary memory is relatively high compared to the matrix and vector operations due to the limited size of vectors and matrices. The following guidelines may help to minimise this overhead in cost.

3.2.4.1 Reuse temporary memory storage

An implementation of Eq. (3.10) starts with the following three steps

`tmp1 =` $\boldsymbol{B}\hat{\boldsymbol{u}}$`;`

tmp2 = $\boldsymbol{D}_{\xi_1}$tmp1;

tmp3 = $\boldsymbol{D}_{\xi_2}$tmp1;

where tmp1, tmp2 and tmp3 are three different temporary arrays used to store the intermediate results. For the computation of the next step, it is possible to reuse the temporary array tmp1, i.e.

tmp1 = $\boldsymbol{G}^{11}$tmp2+$\boldsymbol{G}^{12}$tmp3;

Such a reuse of temporary memory storage not only limits the memory footprint, it also leads to more *cache hits* which may result in a reduction in run-time.

3.2.4.2 Allocate contiguous memory

If you know the total size of all required temporary memory required, it is best to allocate this memory all at once in the beginning of the routine. The different temporary arrays can then be defined as an offset pointer to this *master* array, without the need to make additional allocation calls. This strategy has two advantages:

- There is only one memory allocation call. This will enhance efficiency as it is more the number of calls than the amount of memory allocated that matters.
- All temporary memory is contiguous. Such a locality of data will result in a more efficient memory use, including an increment in cache hits. As said before, this is beneficial for the run-time.

3.2.4.3 Do not allocate temporary memory in the core routines

The operator $\boldsymbol{B}_{\xi_i}$ and $\boldsymbol{D}_{\xi_i}$ and their transposed operators can be considered as the core routines of the Nektar++ library as almost all spectral/*hp* element operators can be constructed as a composition of these operators (see e.g. the weak Laplacian operator). As a result, it is important that they are implemented as efficiently as possible. In Section 5.1.1.2 it is explained that the *backward transformation* operator $\boldsymbol{B}$ is evaluated in two steps and requires temporary memory to store the intermediate result. However, from the previous section, it can be appreciated that allocating this temporary memory inside the function that implements the operator $\boldsymbol{B}$ might not

be optimal, especially because this operator is frequently called from other functions. It is therefore advised to pass some of the contiguous memory storage allocated for the Laplacian operator as an additional argument to the function that implements the operator $\boldsymbol{B}$ (it may be necessary to allocate additional memory storage for this purpose). Another option may be to copy the implementation of the operator $\boldsymbol{B}$ to the Laplacian operator (which also helps to guarantee *inlining*). However, we prefer not to follow this strategy as it will lead to many instances of duplicate code, which somehow violates the idea of a generic conceptual approach (and may lead to problems in case of code improvements or code maintenance).

Chapter 4

A Generic Framework for Time-Stepping Partial Differential Equations

In the development of simulation software into which numerical approximation strategies for solving time-dependent partial differential equations (PDEs) are implemented, the time-stepping method and its implementation typically receive a subordinate role to the modelling and spatial discretisation choices. There exist a myriad of reasons why this partitioning of effort exists and is justified. In part, the Method of Lines (MoL), which is commonly employed to help simplify the discretisation process, focuses one's attention on distilling the partial differential equations down to a collection of coupled ordinary differential equations (ODEs) to which classic time-stepping methods can be applied (see e.g. (Schiesser 1991) for a discussion of the MoL approach). A large amount of effort is thus invested into this distillation process of modelling and spatial discretisation, and the final ODE discretisation stage is often viewed as a straightforward process requiring little concentrated focus. For engineering practices, one then typically begins his testing with an implementation of Euler-Forward – the parent time-stepping method of almost all multi-step and multi-stage schemes. In many applications, the practitioner does not go beyond this point, on the one hand because the Euler-Forward method may be the only

method natively available in the software package (such as e.g. in the OpenFOAM CFD solver (OpenFOAM 2010)), but also arguing that the time-stepping error is sufficiently small compared to modelling, spatial and parameter errors, such that no further effort should be invested into more elaborate time-stepping algorithms. For those applications in which high-order time-stepping is advantageous, practitioners encounter the multi-stage/multi-step divide – whether to use multi-step methods like Adams-Bashforth and Adams-Moulton, which typically require more memory but have an economy of floating-point operations, or to use multi-stage methods like Runge-Kutta (RK), which typically have larger stability regions and require less memory. Whichever selection is made might require further reworking of their simulation software to accommodate either the memory needs or evaluation needs of the family of schemes selected. This serves further to discourage fully exploiting all the advances that have been made in the numerical solution of ODEs and discourages doing verification studies in which the interplay between spatial and temporal discretisation errors (beyond just leading order-of-accuracy statements) are quantified.

The goal of this effort was to develop a generic framework, both in terms of algorithms and software implementations, which allows the user to almost seamlessly switch between various explicit and implicit time-stepping methods. The first challenge we encountered was the question of how to span the multi-stage/multi-step divide. By basing our algorithms on J.C. Butcher's unifying *General Linear Methods* (Burrage & Butcher 1980; Butcher 1987; Butcher 2006), we are able to accommodate a wide range of the time-stepping schemes used in engineering practice, which not only encourages the judicious use of the plethora of different methods that exist, but also facilitates time-discretisation verification studies. However, the ODE concept of general linear methods currently does not encompass the family of implicit-explicit (IMEX) schemes that are often used to time-integrate PDEs as encountered in CFD, see e.g. (Ascher, Ruuth, & Wetton 1995; Ascher, Ruuth, & Spiteri 1997). In order to treat these schemes in a similar way, we have shown that through a small modification, these schemes as well can be formulated as a general linear method.

Although the MoL approach, in principle, abstracts away the spatial discretisation part of the PDE, there are some specific issues arising during this procedure that have a decisive influence on the design of a generic PDE time-stepping framework. In particular, it is the question how to deal with boundary conditions in a generic and computationally efficient manner that forms the second major challenge of this chapter.

Finally, a remark regarding terminology when dealing with time-stepping schemes that are formally explicit from an ODE point of view. Spatially discretising a PDE using a Galerkin approach generally leads to a ODE system which involves the *inversion* of a global system regardless of the fact that we are using an explicit time-stepping scheme. This situation can be referred to as an *indirect* explicit method in contrast to the *direct* explicit method resulting from, for example, a standard finite difference discretisation. Also note that the framework we will present is only valid for implicit schemes in which the stage values can be computed in a decoupled fashion. Such decoupled schemes include all implicit multi-step schemes and the diagonally implicit multi-stage schemes such as the *DIRK* schemes. Fully implicit multi-stage methods, which are rarely adopted in engineering practice, do not fit into the presented framework.

Design Considerations Based upon the aforementioned motivations, we set the following as the objectives of our time-stepping framework:

- It should allow the user to select its preferred method in an unbiased fashion.
- It should facilitate both explicit and implicit time-stepping.
- It should facilitate both multi-step and multi-stage schemes.
- It should allow the user to implement more elaborate partitioning schemes, e.g. Implicit-Explicit (IMEX) schemes.
- It should be designed anticipating that the Method of Lines has been used on a PDE to yield a system of coupled ODEs. Therefore the framework should be able to accommodate both static and time-dependent boundary conditions.

- It should work independent of the spatial discretisation choice (i.e. it should work with continuous Galerkin and discontinuous Galerkin methods, as well as with finite difference and finite volume methods).
- It should provide an efficient solution for time-stepping PDEs, i.e. the computational cost should be comparable to scheme-specific implementations.

Outline In this chapter, we document the objectives of our framework, provide a brief overview of general linear methods and explain how we design and implement a software solution written in an object-oriented language (OOL) that meets our objectives. The chapter is organized in sections as follows. We begin in Section 4.1 by presenting Butcher's idea of general linear methods. We provide examples to help the reader appreciate how this framework accommodates the commonly used multi-step and multi-stage schemes and we show how IMEX schemes can also be worked into this framework. Section 4.2 describes the design, algorithms and implementation of the ODE solving framework. In Section 4.3, we then present how this ODE framework can be modified into a generic PDE time-stepping framework meeting the objectives. Therefore, we introduce the Method of Lines decomposition of a model problem and we explain how to deal with time-dependent boundary conditions in a generic and computationally efficient way. Finally in Section 4.4, we demonstrate the capabilities of the presented framework by presenting some examples.

4.1 General Linear Methods

General linear methods (GLM) have emerged as an effort to connect two main types of time integration schemes: linear multi-step methods and linear multi-stage methods. Linear multi-step methods, such as the Adams family of schemes, use the collection of r input parameters from the previous time-levels to obtain the solution at the next time-level. On the other hand, linear multi-stage methods such as Runge-Kutta methods, approximate the solution at the new time-level by linearly combining the solution at s intermediate stages.

To begin, the standard *initial value problem* in autonomous form is represented by the ODE,

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \tag{4.1}$$

where $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$. The n^{th} step of the general linear method comprised of r steps and s stages is then formulated as (Burrage & Butcher 1980):

$$\boldsymbol{Y}_i = \Delta t \sum_{j=1}^{s} a_{ij} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \leq i \leq s, \tag{4.2a}$$

$$\boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \leq i \leq r, \tag{4.2b}$$

where $\boldsymbol{Y}_i$ are called the stage values and $\boldsymbol{F}_i$ are called the stage derivatives. Both quantities are related by the differential equation:

$$\boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i). \tag{4.2c}$$

The matrices $A = [a_{ij}]$, $U = [u_{ij}]$, $B = [b_{ij}]$, $V = [v_{ij}]$ are characteristic of a specific method, and as a result, each scheme can be uniquely defined by the partitioned $(s + r) \times (s + r)$ matrix

$$\begin{bmatrix} A & U \\ B & V \end{bmatrix}. \tag{4.3}$$

For a more concise notation, it is convenient to define the vectors $\boldsymbol{Y}, \boldsymbol{F} \in \mathbb{R}^{sN}$ and $\boldsymbol{y}_i^{[n-1]}, \boldsymbol{y}_i^{[n]} \in \mathbb{R}^{rN}$ as follows:

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y}_1 \\ \boldsymbol{Y}_2 \\ \vdots \\ \boldsymbol{Y}_s \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} \boldsymbol{F}_1 \\ \boldsymbol{F}_2 \\ \vdots \\ \boldsymbol{F}_s \end{bmatrix}, \quad \boldsymbol{y}^{[n-1]} = \begin{bmatrix} \boldsymbol{y}_1^{[n-1]} \\ \boldsymbol{y}_2^{[n-1]} \\ \vdots \\ \boldsymbol{y}_r^{[n-1]} \end{bmatrix}, \text{ and } \boldsymbol{y}^{[n]} = \begin{bmatrix} \boldsymbol{y}_1^{[n]} \\ \boldsymbol{y}_2^{[n]} \\ \vdots \\ \boldsymbol{y}_r^{[n]} \end{bmatrix}. \tag{4.4}$$

Using these vectors, it is possible to write Eq. (4.2a) and Eq. (4.2b) in the form

$$\begin{bmatrix} \boldsymbol{Y} \\ \boldsymbol{y}^{[n]} \end{bmatrix} = \begin{bmatrix} A \otimes I_N & U \otimes I_N \\ B \otimes I_N & V \otimes I_N \end{bmatrix} \begin{bmatrix} \Delta t \boldsymbol{F} \\ \boldsymbol{y}^{[n-1]} \end{bmatrix}, \tag{4.5}$$

where I_N is the identity matrix of dimension $N \times N$ and $\otimes$ denotes the Kronecker product. Note that it is the first element of the input vector $\boldsymbol{y}^{[n-1]}$ and output

vector $\boldsymbol{y}^{[n]}$ which represents the solution at the corresponding time-level, i.e. $\boldsymbol{y}_1^{[n]} = \boldsymbol{y}_n = \boldsymbol{y}(t_0 + n\Delta t)$. The other subvectors $\boldsymbol{y}_i^{[n]}$ $(2 \leq i \leq r)$ refer to the approximation of an auxiliary set of values inherent to the scheme. These values, in general, are comprised of either solutions or derivatives at earlier time-levels or a combination hereof.

4.1.1 Common multi-stage methods

Since multi-stage methods consist only of a single step with many stages, they can be represented as a general linear method with $r = 1$. It is sufficient to write $U = [\ 1 \quad 1 \ \cdots \ 1\ ]^\top$, $V = [1]$ and to set the coefficient matrices A and B to the matrix A and the single row $b^\top$ of the corresponding *Butcher tableau* (Butcher 2006) respectively. For example, the classic fourth-order Runge-Kutta method with Butcher tableau

$$\begin{array}{c|c} c & A \\ \hline & b^\top \end{array} = \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}, \tag{4.6}$$

has the following GLM representation

$$\left[\begin{array}{cc} A & U \\ B & V \end{array}\right] = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ \hline \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & 1 \end{array}\right]. \tag{4.7}$$

4.1.2 Common multi-step methods

In contrast to multi-stage methods, multi-step methods have a single stage, but the solution at the new time-level is computed as a linear combination of information at the r previous time-levels. Linear multi-step methods can be formulated to satisfy

the relation

$$\boldsymbol{y}_n = \sum_{i=1}^{r} \alpha_i \boldsymbol{y}_{n-i} + \Delta t \sum_{i=0}^{r} \beta_i \boldsymbol{F}_{n-i}. \tag{4.8}$$

This corresponds to the general linear method with input and output

$$\boldsymbol{y}^{[n-1]} = \begin{bmatrix} \boldsymbol{y}_{n-1} \\ \boldsymbol{y}_{n-2} \\ \vdots \\ \boldsymbol{y}_{n-r} \\ \hdashline \Delta t \boldsymbol{F}_{n-1} \\ \Delta t \boldsymbol{F}_{n-2} \\ \vdots \\ \Delta t \boldsymbol{F}_{n-r} \end{bmatrix}, \quad \boldsymbol{y}^{[n]} = \begin{bmatrix} \boldsymbol{y}_{n} \\ \boldsymbol{y}_{n-1} \\ \vdots \\ \boldsymbol{y}_{n-r+1} \\ \hdashline \Delta t \boldsymbol{F}_{n} \\ \Delta t \boldsymbol{F}_{n-1} \\ \vdots \\ \Delta t \boldsymbol{F}_{n-r+1} \end{bmatrix}, \tag{4.9}$$

and the partitioned coefficient matrix $\left[\begin{smallmatrix} A & U \\ B & V \end{smallmatrix}\right]$ defined as

$$\left[\begin{array}{c|cccccc:cccccc} \beta_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{r-1} & \alpha_r & \beta_1 & \beta_2 & \cdots & \beta_{r-1} & \beta_r \\ \hline \beta_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{r-1} & \alpha_r & \beta_1 & \beta_2 & \cdots & \beta_{r-1} & \beta_r \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \hdashline 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1 & 0 \end{array}\right]. \tag{4.10}$$

Note that in the vectors and matrices above, the solid lines denote the demarcation of the matrices A, B, U and V whereas the dotted lines merely help to highlight the typical structure of a linear multi-step method. As an example of a multi-step scheme consider the well-known third-order Adams-Bashforth scheme

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{23}{12} \boldsymbol{f}(\boldsymbol{y}_{n-1}) - \frac{4}{3} \boldsymbol{f}(\boldsymbol{y}_{n-2}) + \frac{5}{12} \boldsymbol{f}(\boldsymbol{y}_{n-3}) \right), \tag{4.11}$$

which has the following GLM representation

$$\left[\begin{array}{cc} A & U \\ B & V \end{array}\right] = \left[\begin{array}{c|cccc} 0 & 1 & \frac{23}{12} & -\frac{4}{3} & \frac{5}{12} \\ \hline 0 & 1 & \frac{23}{12} & -\frac{4}{3} & \frac{5}{12} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right]. \tag{4.12}$$

4.1.3 Beyond common multi-step or multi-stage methods

The general linear methods framework also encompasses methods that do not fit under the conventional Runge-Kutta or linear multi-step headings. This includes, for example, the cyclic composite method of (Donelson & Hansen 1971). In addition, the general linear structure of the GLM in itself gave rise to the development of new numerical methods. An example of one such class of methods is the class of *Almost Runge-Kutta Methods* (Butcher 1997). To appreciate its typical combined multi-stage multi-step character consider the following third-order scheme due to (Rattenbury 2005):

$$\left[\begin{array}{cc} A & U \\ B & V \end{array}\right] = \left[\begin{array}{ccc|ccc} 0 & 0 & 0 & 1 & \frac{1}{3} & \frac{1}{18} \\ \frac{1}{2} & 0 & 0 & 1 & \frac{1}{6} & \frac{1}{18} \\ 0 & \frac{3}{4} & 0 & 1 & \frac{1}{4} & 0 \\ \hline 0 & \frac{3}{4} & 0 & 1 & \frac{1}{4} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 3 & -3 & 2 & 0 & -2 & 0 \end{array}\right]. \tag{4.13}$$

4.1.4 Implicit-explicit general linear methods

In this section, we extend the idea of GLM to accommodate in addition implicit-explicit (IMEX) schemes. IMEX schemes (Ascher, Ruuth, & Wetton 1995; Ascher, Ruuth, & Spiteri 1997) were introduced to time-integrate ODEs of the form

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}) + \boldsymbol{g}(\boldsymbol{y}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \tag{4.14}$$

where $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$ typically is a non-linear function and $\boldsymbol{g} : \mathbb{R}^N \to \mathbb{R}^N$ is a stiff term (or where $\boldsymbol{f}$ and $\boldsymbol{g}$ have disparate time-scales). The idea behind IMEX methods is to combine two different type of schemes: one would like to use an implicit scheme for the stiff term in order to avoid an excessively small time-step. At the same time, explicit integration of the non-linear term is preferred to avoid its expensive inversion.

Following the same underlying idea as discussed in the previous sections, IMEX linear multi-step schemes (Ascher, Ruuth, & Wetton 1995) and IMEX Runge-Kutta schemes (Ascher, Ruuth, & Spiteri 1997) can be unified into an IMEX general linear method formulation, i.e.

$$\boldsymbol{Y}_i = \Delta t \sum_{j=1}^{s} a_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} a_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \le i \le s, \tag{4.15a}$$

$$\boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \le i \le r, \tag{4.15b}$$

where the stage derivatives $\boldsymbol{F}_i$ and $\boldsymbol{G}_i$ are defined as

$$\boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i), \quad \boldsymbol{G}_i = \boldsymbol{g}(\boldsymbol{Y}_i), \tag{4.15c}$$

and where the superscripts IM and EX are used to denote implicit and explicit respectively. Adopting a matrix formulation similar to that shown in Eq. (4.5), this can be written in the form

$$\left[\begin{array}{c} \boldsymbol{Y} \\ \hline \boldsymbol{y}^{[n]} \end{array}\right] = \left[\begin{array}{c:c|c} A^{\mathrm{IM}} \otimes I_N & A^{\mathrm{EX}} \otimes I_N & U \otimes I_N \\ \hline B^{\mathrm{IM}} \otimes I_N & B^{\mathrm{EX}} \otimes I_N & V \otimes I_N \end{array}\right] \left[\begin{array}{c} \Delta t \boldsymbol{G} \\ \Delta t \boldsymbol{F} \\ \hline \boldsymbol{y}^{[n-1]} \end{array}\right]. \tag{4.16}$$

To further illustrate the formulation of IMEX schemes as a GLM, consider the following examples. The first-order Backward Euler/Forward Euler IMEX scheme,

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\boldsymbol{g}(\boldsymbol{y}_n) + \boldsymbol{f}(\boldsymbol{y}_{n-1}) \right), \tag{4.17}$$

can be written as a general linear method as

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cc} 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right] \quad \text{with} \quad \boldsymbol{y}^{[n]} = \begin{bmatrix} \boldsymbol{y}_n \\ \Delta t \boldsymbol{F}_n \end{bmatrix}. \tag{4.18}$$

The second-order Crank-Nicholson/Adams-Bashforth linear multi-step scheme,

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{1}{2}\boldsymbol{g}(\boldsymbol{y}_n) + \frac{1}{2}\boldsymbol{g}(\boldsymbol{y}_{n-1}) + \frac{3}{2}\boldsymbol{f}(\boldsymbol{y}_{n-1}) - \frac{1}{2}\boldsymbol{f}(\boldsymbol{y}_{n-1}) \right), \tag{4.19}$$

can be represented as

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cccc} \frac{1}{2} & 0 & 1 & \frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ \hline \frac{1}{2} & 0 & 1 & \frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right], \tag{4.20}$$

with input/output vector

$$\boldsymbol{y}^{[n]} = \begin{bmatrix} \boldsymbol{y}_n \\ \Delta t \boldsymbol{G}_n \\ \Delta t \boldsymbol{F}_n \\ \Delta t \boldsymbol{F}_{n-1} \end{bmatrix}. \tag{4.21}$$

The third-order $(2,3,3)$ IMEX Runge-Kutta scheme (see (Ascher, Ruuth, & Spiteri 1997)) is represented by the partitioned coefficient matrix where $\gamma = (3+\sqrt{3})/6$:

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{ccc:ccc|c} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \gamma & 0 & \gamma & 0 & 0 & 1 \\ 0 & 1-2\gamma & \gamma & \gamma-1 & 2(1-\gamma) & 0 & 1 \\ \hline 0 & \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} & 1 \end{array}\right]. \tag{4.22}$$

4.2 A generic ODE solving framework

Just as Butcher's general linear methods provide a general framework to study the basic properties such as consistency, stability and convergence of different families of numerical methods for ODEs, it can also serve as a starting point for a unified numerical implementation. For maximum generality we base our implementation on the IMEX-GLM formulation described in Section 4.1.4: for purely explicit methods we simply define A^{IM}, B^{IM} as well as $\boldsymbol{g}(\boldsymbol{y})$ equal to zero. For purely implicit schemes we analogously set A^{EX}, B^{EX} and $\boldsymbol{f}(\boldsymbol{y})$ to be zero.

4.2.1 Evaluation of general linear methods

Inspecting Eq. (4.15) it can be appreciated that a single step from level $n-1$ to n for an IMEX-GLM formulation can be evaluated through the following algorithm:

input : the vector $\boldsymbol{y}^{[n-1]}$

output: the vector $\boldsymbol{y}^{[n]}$

// Calculate stage values $\boldsymbol{Y}_i$ and the stage derivatives $\boldsymbol{F}_i$ and $\boldsymbol{G}_i$

for $i = 1$ **to** s **do**

// calculate the temporary variable $\boldsymbol{x}_i$

(A1.1) $\quad \boldsymbol{x}_i = \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}$

// calculate the stage value $\boldsymbol{Y}_i$

(A1.2) $\quad$ solve $\left(\boldsymbol{Y}_i - a_{ii}^{\mathrm{IM}} \Delta t \boldsymbol{g}(\boldsymbol{Y}_i) \right) = \boldsymbol{x}_i$

// calculate the explicit stage derivative $\boldsymbol{F}_i$

(A1.3) $\quad \boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i)$

// calculate the implicit stage derivative $\boldsymbol{G}_i$

(A1.4) $\quad \boldsymbol{G}_i = \boldsymbol{g}(\boldsymbol{Y}_i) = \frac{1}{a_{ii}^{\mathrm{IM}} \Delta t} \left(\boldsymbol{Y}_i - \boldsymbol{x}_i \right)$

end

// Calculate the output vector $\boldsymbol{y}^{[n]}$

for $i = 1$ **to** r **do**

// calculate $\boldsymbol{y}_i^{[n]}$

(A1.5) $\quad \boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}$

end

Algorithm 1: A GLM-based ODE solving algorithm.

Here we first observe that the algorithm, by virtue of the GLM framework, is independent of the actual numerical scheme used – only the *values* of the coefficients a and b change for different methods. Further, if we are using a purely explicit scheme then $a_{ii}^{\mathrm{IM}} = 0$ and the stage value is equal to the the temporary value computed in step (A1.1), i.e. step (A1.2) is greatly simplified to $\boldsymbol{Y}_i = \boldsymbol{x}_i$. It is also worth noting that the stage derivative $\boldsymbol{G}_i = \boldsymbol{g}(\boldsymbol{Y}_i)$ in (A1.4) need not be explicitly evaluated, but is given by already computed values, as seen by reordering Eq. (A1.2). Indeed, steps (A1.2) and (A1.3) are the only instances in the algorithm where specific information

from the ODE is required, all other steps in the algorithm simply involve linear combinations of precomputed information. These two steps are to be considered *external* parts representing the ODE rather than being a part of the numerical GLM algorithm. We thus need to define the following external functions:

- If $A^{\mathrm{IM}} \neq 0$ we must supply a routine for solving a system of form (A1.2),

$$(\boldsymbol{Y} - \lambda \boldsymbol{g}(\boldsymbol{Y})) = \boldsymbol{x}, \tag{4.23}$$

  for $\boldsymbol{Y} \in \mathbb{R}^N$, given as input the vector $\boldsymbol{x} \in \mathbb{R}^N$ and the scalar $\lambda \in \mathbb{R}$. $\boldsymbol{\mathcal{I}} : \mathbb{R}^N \to \mathbb{R}^N$ denotes the identity function and $\boldsymbol{g} : \mathbb{R}^N \to \mathbb{R}^N$ is the function prescribing the terms of the ODE that are to be implicitly evaluated. In general, the solution or fixed point of this system can be found through root finding algorithms. In the case $\boldsymbol{g}$ is a linear operator, one may opt for a direct solution method to solve this system through the inverse operator $(\boldsymbol{\mathcal{I}} - \lambda \boldsymbol{g})^{-1}$.

- If $A^{\mathrm{EX}} \neq 0$ we must also supply a routine for evaluating (A1.3), i.e. a function $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$ that maps the stage values to the stage derivatives for the terms of the ODE that are explicitly evaluated

$$\boldsymbol{F} = \boldsymbol{f}(\boldsymbol{Y}). \tag{4.24}$$

The external functions are expected to be provided by the user. The *decoupling* of the external components from the GLM algorithm naturally leads to a level of abstraction allowing a generic object-oriented implementation, in a programming language such as C++, to be discussed in the next section.

4.2.2 Encapsulation of key concepts

As outlined in the introduction, it is our goal to implement an ODE solving toolbox where switching between numerical schemes is as simple as changing an input parameter. To accomplish this, we have encapsulated the key concepts observed in the previous section into a set of C++ type classes. It is not our intention to necessarily

advocate using only C++ but rather to highlight how any object-oriented language (OOL) could be used to encapsulate the concepts. We acknowledge that many OOL exist which could be used for this implementation stage.

An overview of these classes is depicted in Fig. 4.1.

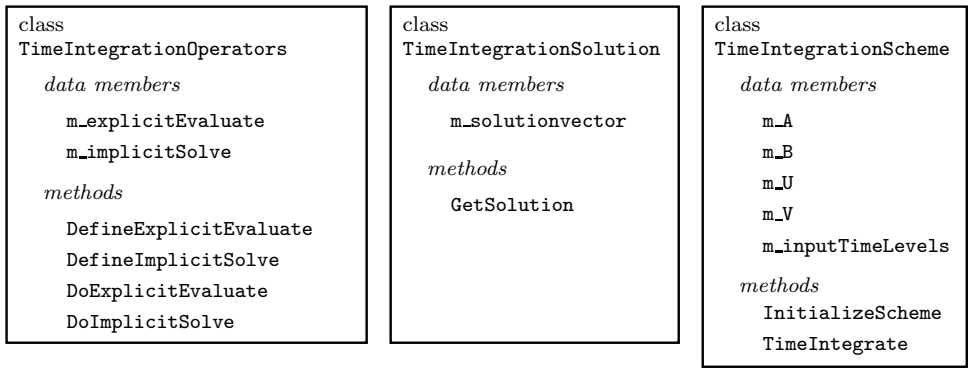


Figure 4.1: Overview of the classes in the implementation of the generic ODE solving framework.

4.2.2.1 The class TimeIntegrationOperators

This class provides a general interface to the *external* components (see Section 4.2.1) needed for time marching. As a result, this class can be seen as the abstraction of the ODE. As data members, it contains two objects which can be thought of as *function pointers*: `m_explicitEvaluate` should point to the implementation of (4.24) and `m_implicitSolve` should point to the implementation of solving system (4.23). The function pointers can be linked to these implementations by means of the methods `DefineExplicitEvaluate` and `DefineImplicitSolve`. Note that it is up to the user to implement and provide these functions. The encapsulation of these functions into another class is needed to ensure that both these functions can be accessed from within the time-stepping algorithms in a unified fashion, independent of which ODE is being solved. Therefore, the class `TimeIntegrationOperators` also contains the methods `DoExplicitEvaluate` and `DoImplicitSolve` for internal use.

4.2.2.2 The class TimeIntegrationSolution

This class is the abstraction of the vector $\boldsymbol{y}^{[n]}$ as defined in Eq. (4.4). One can think of it as an array of arrays. The user can obtain its first entry $\boldsymbol{y}_1^{[n]} = \boldsymbol{y}_n$ representing the approximate solution at time-level n, by means of the method `GetSolution`.

4.2.2.3 The class TimeIntegrationScheme

This class can be considered as the main class as it is the abstraction of a general linear method. As each scheme is uniquely defined by the partitioned coefficient matrix (4.3), the sub-matrices A, B, U and V are core data members of this class, implemented respectively as `m_A`, `m_B`, `m_U` and `m_V`. In addition, this class contains the data member `m_inputTimeLevels` which reflect the structure of the input/output vector $\boldsymbol{y}^{[n]}$ associated to the scheme. Based upon the fact that all input vectors can be ordered such that the stage values are listed first before the explicit stage derivatives and the implicit stage derivatives, `m_inputTimeLevels` can be seen as an array existing of three parts that indicate the time-level at which the values/derivatives are evaluated. As an example, consider the second-order Crank-Nicholson/Adams-Bashforth scheme with input vector $\boldsymbol{y}^{[n]}$ as defined in Eq. (4.20). The data member `m_inputTimeLevels` is then defined as

$$\texttt{m\_inputTimeLevels} = \left[\begin{array}{c} 0 \\ \hline 0 \\ \hline 0 \\ 1 \end{array}\right]. \tag{4.25}$$

Furthermore, this class is equipped with the two methods needed for the actual time-marching. The function `InitializeScheme` converts the initial value $\boldsymbol{y}_0$ in an object of the class `TimeIntegrationSolution`. This object is then going to be advanced in time using the method `TimeIntegrate`. This is the function which actually implements the GLM algorithm in Section 4.2.1 and hence integrates the ODE for a single time-step. Note that internally, this method calls the functions `DoExplicitEvaluate` and `DoImplicitSolve` of the class `TimeIntegrationOperators` in order to evaluate Eq. (4.24) and solve Eq. (4.23) respectively.

4.2.3 Use of the framework

As a first step, it is up to the user to provide a proper implementation of the functionality described by Eqs. (4.23) and (4.24). Both these functions should be implemented according to the prototypes below. The function names `ExplicitEvaluate` and `ImplicitSolve` are merely illustrative; the user is free to choose other names.

```
double* ExplicitEvaluate(double* x)
{
    ... // Implemented by the user
}

double* ImplicitSolve(double* x, double lambda)
{
    ... // Implemented by the user
}
```

These functions, together with the classes of the toolbox, can then be used to numerically solve the ODE. A typical example of how this can be implemented is shown below:

```
TimeIntegrationOperators  ode;
TimeIntegrationSolution   y_n;
TimeIntegrationScheme     scheme;

ode.DefineExplicitEvaluate(ExplicitEvaluate);
ode.DefineImplicitSolve(ImplicitSolve);

scheme = TimeIntegrationScheme(FORWARD_EULER);

y_n = scheme->InitializeScheme(dt,y_0,ode);
```

```
for(n = 0; n < nsteps; ++n)
{
    scheme->TimeIntegrate(dt,y_n,ode);
}
```

In this particular example, the constructor call `scheme = TimeIntegrationScheme (FORWARD_EULER)` loads the object with the coefficient matrices of the forward Euler scheme. However, none of the implementation required by the user changes when selecting a more advanced time-stepping method. Other schemes can simply be loaded by changing the input argument (e.g. from `FORWARD_EULER` to `CLASSICAL_RK_4`). This demonstrates that the presented framework indeed does allow the user to numerically solve an ODE in a unified fashion, independent of the chosen scheme.

4.2.3.1 Initiating multi-step schemes

For multi-step schemes, a slight modification is required to properly start-up the system. This can be understood by considering the following example for the third-order Adams-Bashforth scheme.

```
scheme          = TimeIntegrationScheme(ADAMS_BASHFORTH_ORDER3);
startup_scheme1 = TimeIntegrationScheme(FORWARD_EULER);
startup_scheme2 = TimeIntegrationScheme(ADAMS_BASHFORTH_ORDER2);

y_n = scheme->InitializeScheme(dt,y_0,ode);

startup_scheme1->TimeIntegrate(dt,y_n,ode);
startup_scheme2->TimeIntegrate(dt,y_n,ode);
for(n = 2; n < nsteps; ++n)
{
    scheme->TimeIntegrate(dt,y_n,ode);
}
```

Underneath, this starting-up procedure is founded upon the data member `m_inputTimeLevels`. When making the call `startup_scheme1->TimeIntegrate(dt,y_n, ode)`, the `TimeIntegrate` routine recognises that the input vector `y_n` is initialised according to another scheme. It is therefore going first to construct an input vector according to the start-up scheme, and it will map the information from the vector `y_n` to the newly constructed input vector, thereby making use of the data member `m_inputTimeLevels`. If the start-up scheme requires information in its input vector that is not available in the provided input vector `y_n` it will simply assume zero for these stage values or derivatives. Once the solution is advanced in time for a single time-step using the start-up scheme, the output vector is mapped back to the vector `y_n`, again making use of the information in `m_inputTimeLevels`.

4.3 Time-dependent partial differential equations

Ordinary differential equations are generally used to model *initial value problems*. However, many physical processes can be regarded as *initial boundary value problems* which are described by partial differential equations. A first step in solving time-dependent PDEs consists of reducing the PDE to a system of ODEs through the Method of Lines approach. We will show that this procedure, which involves the discretisation of the spatial dimensions, introduces some typical issues which prevent the straightforward application of the ODE framework discussed before. We distinguish the following issues:

- strongly enforced essential boundary conditions, and
- computational efficiency.

To facilitate the discussion we will use the scalar advection-diffusion equation as an illustrative example throughout this section. It is given by

$$\frac{\partial u}{\partial t} + \nabla \cdot \boldsymbol{F}(u) = \nabla^2 u, \qquad \text{in } \Omega \times [0, \infty), \tag{4.26a}$$

$$u(\boldsymbol{x}, t) = g_D(\boldsymbol{x}, t), \qquad \text{on } \partial\Omega_D \times [0, \infty), \tag{4.26b}$$

$$\frac{\partial u}{\partial n}(\boldsymbol{x}, t) = g_N(\boldsymbol{x}, t) \cdot \boldsymbol{n}, \qquad \text{on } \partial\Omega_N \times [0, \infty), \tag{4.26c}$$

$$u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}), \qquad \text{in } \Omega, \tag{4.26d}$$

where Ω is a bounded domain of $\mathbb{R}^d$ with boundary $\partial\Omega = \partial\Omega_D \bigcup \partial\Omega_N$ and $\boldsymbol{n}$ denotes the outward normal to the boundary $\partial\Omega$. Furthermore, we will abbreviate the advection term as $f(u) = -\nabla \cdot \boldsymbol{F}(u)$ in the following sections.

4.3.1 The Method of Lines

We start with a spectral/*hp* element approach to reduce the advection-diffusion equation (4.26) to a system of ODEs through the Method of Lines. Following the standard Galerkin formulation we multiply Eq. (4.26a) by a smooth test function $v(\boldsymbol{x})$, which by definition is zero on all Dirichlet boundaries. Integrating over the entire spatial domain leads to the following variational formulation: Find $u \in \mathcal{U}$ such that

$$\int_\Omega v \frac{\partial u}{\partial t} d\boldsymbol{x} - \int_\Omega v f(u) d\boldsymbol{x} = \int_\Omega v \nabla^2 u d\boldsymbol{x}, \quad \forall v \in \mathcal{V}, \tag{4.27}$$

where $\mathcal{U}$ and $\mathcal{V}$ are suitably chosen trial and test spaces respectively. We obtain the weak form by applying the divergence theorem to the right-hand-side term yielding: Find $u \in \mathcal{U}$ such that

$$\int_\Omega v \frac{\partial u}{\partial t} d\boldsymbol{x} - \int_\Omega v f(u) d\boldsymbol{x} = -\int_\Omega \nabla v \cdot \nabla u d\boldsymbol{x} + \int_{\partial\Omega} v \nabla u \cdot \boldsymbol{n} d\boldsymbol{x}, \quad \forall v \in \mathcal{V}. \tag{4.28}$$

As $v(\partial\Omega_D)$ is equal to zero, only Neumann conditions will give contributions to the boundary integral, and we enforce the conditions weakly through substituting $\nabla u = \boldsymbol{g}_N$ in the boundary integral. In order to impose Dirichlet boundary conditions we adopt a lifting strategy where the solution is decomposed into a known function, u^D and an unknown homogeneous function u^H, i.e.

$$u(\boldsymbol{x}, t) = u^H(\boldsymbol{x}, t) + u^D(\boldsymbol{x}, t). \tag{4.29}$$

Here u^D is satisfying the Dirichlet boundary conditions, $u^D(\partial\Omega_D) = g_D$, and the homogeneous function is equal to zero on the Dirichlet boundary, $u^H(\partial\Omega_D) = 0$. The weak form (4.28) can then be formulated as: Find $u^D \in \mathcal{U}_0$ such that,

$$\int_\Omega v\frac{\partial(u^H + u^D)}{\partial t}d\boldsymbol{x} - \int_\Omega vf(u^H + u^D)d\boldsymbol{x} = -\int_\Omega \nabla v \cdot (\nabla u^H + \nabla u^D)d\boldsymbol{x} + \int_{\partial\Omega_N} v\boldsymbol{g}_N \cdot \boldsymbol{n}d\boldsymbol{x}, \quad \forall v \in \mathcal{V}. \tag{4.30}$$

Following a finite element discretisation procedure, the solution is expanded in terms of a globally C^0-continuous expansion basis Φ_i that spans the finite dimensional solution space $\mathcal{U}^\delta$. We also decompose this expansion basis into the homogeneous basis functions Φ_i^H and the basis functions Φ_i^D having support on the Dirichlet boundary such that

$$u^\delta(\boldsymbol{x}, t) = \sum_{i\in\mathcal{N}^H} \Phi_i^H(\boldsymbol{x})\hat{u}_i^H(t) + \sum_{i\in\mathcal{N}^D} \Phi_i^D(\boldsymbol{x})\hat{u}_i^D(t). \tag{4.31}$$

Finally, employing the same expansion basis Φ_i^H to span the test space $\mathcal{V}$, Eq. (4.30) leads to the semi-discrete system of ODEs

$$\begin{bmatrix} \boldsymbol{M}^{HD} & \boldsymbol{M}^{HH} \end{bmatrix} \frac{d}{dt}\begin{bmatrix} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{bmatrix} = -\begin{bmatrix} \boldsymbol{L}^{HD} & \boldsymbol{L}^{HH} \end{bmatrix}\begin{bmatrix} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{bmatrix} + \boldsymbol{\Gamma}^H + \hat{\boldsymbol{f}}^H \tag{4.32}$$

where

$$\begin{aligned}
\boldsymbol{M}^{HD}[i][j] &= \int_\Omega \Phi_i^H\Phi_j^D d\boldsymbol{x} && i \in \mathcal{N}^H, j \in \mathcal{N}^D,\\
\boldsymbol{L}^{HD}[i][j] &= \int_\Omega \nabla\Phi_i^H \cdot \nabla\Phi_j^D d\boldsymbol{x} && i \in \mathcal{N}^H, j \in \mathcal{N}^D,\\
\hat{\boldsymbol{f}}^H[i] &= \int_\Omega \Phi_i^H f(u)d\boldsymbol{x} && i \in \mathcal{N}^H\\
\boldsymbol{\Gamma}^H[i] &= \int_{\partial\Omega_N} \Phi_i^H\boldsymbol{g}_N \cdot \boldsymbol{n}d\boldsymbol{x} && i \in \mathcal{N}^H.
\end{aligned}$$

This can be rewritten in terms of the unknown variable $\hat{\boldsymbol{u}}^H$ as

$$\frac{d\hat{\boldsymbol{u}}^H}{dt} = \left(\boldsymbol{M}^{HH}\right)^{-1}\left\{-\begin{bmatrix} \boldsymbol{L}^{HD} & \boldsymbol{L}^{HH} \end{bmatrix}\begin{bmatrix} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{bmatrix} + \boldsymbol{\Gamma}^H + \hat{\boldsymbol{f}}^H - \boldsymbol{M}^{HD}\frac{d\hat{\boldsymbol{u}}^D}{dt}\right\}, \tag{4.33}$$

which, in the absence of Dirichlet boundary conditions, simplifies to

$$\frac{d\hat{\boldsymbol{u}}}{dt} = -\boldsymbol{M}^{-1}\left(\boldsymbol{L}\hat{\boldsymbol{u}} - \boldsymbol{\Gamma}\right) + \boldsymbol{M}^{-1}\hat{\boldsymbol{f}} \tag{4.34}$$

4.3.2 Use of the ODE framework for time integrating PDEs

At first sight, it may seem feasible to apply to ODE solving framework of Section 4.2 to time-integrate Eq. (4.33) (or Eq. (4.34)). However, this straightforward approach appears to lead to two different problems.

4.3.2.1 Computational efficiency

Considering Eq. (4.34) in the context of the IMEX algorithm of the ODE framework (Algorithm 1), it can be appreciated that for the explicit advection term, step (A1.4) requires the calculation of the term

$$\boldsymbol{M}^{-1}\hat{\boldsymbol{f}}, \tag{4.35}$$

while for the implicit diffusion term, step (A1.2) would require solving a system of the form

$$\left(\boldsymbol{I} + \Delta t \boldsymbol{M}^{-1}\boldsymbol{L}\right)\hat{\boldsymbol{u}} = \hat{\boldsymbol{x}}. \tag{4.36}$$

It appears that next to the implicit term, the explicit term now also requires a global matrix inverse due to $\boldsymbol{M}^{-1}$. This means that the generic ODE time-stepping algorithm would require two global matrix inverses at every timestep/timestage. For comparison, let us consider the (single-stage) first-order Backward Euler/Forward Euler IMEX scheme given by Eq. (4.17). A scheme-specific implementation of this method (that is, not making use of the proposed framework) can integrate Eq. (4.34) for a single time-step as

$$\hat{\boldsymbol{u}}_n = \left(\boldsymbol{M} + \Delta t \boldsymbol{L}\right)^{-1}\left(\boldsymbol{M}\hat{\boldsymbol{u}}_{n-1} + \Delta t \hat{\boldsymbol{f}}_{n-1} + \Delta t \boldsymbol{\Gamma}_n\right). \tag{4.37}$$

Clearly, this only involves a single global matrix inversion, i.e. due to $\left(\boldsymbol{M} + \Delta t \boldsymbol{L}\right)^{-1}$. Such global matrix inversions can in general be assumed to be the critical cost of the time-integration process as they typically –especially for three-dimensional simulations– require an iterative solution method which induce a far bigger cost than the other *forward* operations. Because of this substantial performance penalty, using the ODE framework to time-integrate PDEs can be considered an impractical solution.

4.3.2.2 Time-dependent boundary conditions

The second complication arises from the term $\frac{d\hat{\boldsymbol{u}}^D}{dt}$ in Eq. (4.33) which is due to a strong imposition of the Dirichlet boundary conditions. Although the value $\hat{\boldsymbol{u}}^D(t)$ of the Dirichlet boundary conditions is given for arbitrary t (or can be computed based upon Eq. (4.26b), see (Karniadakis & Sherwin 2005)), no prescription for its time rate-of-change $\frac{d\hat{\boldsymbol{u}}^D}{dt}$ is available in general. This prevents a straightforward application of the presented ODE framework.

4.3.3 A generic PDE time-stepping framework

In order to alleviate both the issues of efficiency and time-dependent boundary conditions, we propose a modified framework designed to time-integrate PDEs in a generic *and* efficient manner. The new framework is largely founded on the fact that a spectral/*hp* element approximation $u^\delta(\boldsymbol{x},t)$ can be described not only by a set of global degrees of freedom $\hat{\boldsymbol{u}}$ (in *coefficient space*), but also by a set of nodal values $\boldsymbol{u}$ (in *physical space*). These nodal values represent the spectral/*hp* solution at a set of quadrature points $\boldsymbol{x}_i$ (or *collocation points*), such that they can be related to the global coefficients as

$$\boldsymbol{u}[i] = u^\delta(\boldsymbol{x},t) = \sum_{j\in\mathcal{N}} \Phi_j(\boldsymbol{x}_i)\hat{u}_i(t), \tag{4.38}$$

which, in matrix notation, can be written as $\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}$, where $\boldsymbol{B}[i][j] = \Phi_j(\boldsymbol{x}_i)$. This can be recognised as the global variant of the elemental *backward transformation* as defined in Section 2.2.4.3. In case of a lifted Dirichlet solution, this becomes

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{B}^D & \boldsymbol{B}^H \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{bmatrix}, \tag{4.39}$$

with $\boldsymbol{B}^D[i][j] = \Phi_j^D(\boldsymbol{x}_i)$ and $\boldsymbol{B}^H[i][j] = \Phi_j^H(\boldsymbol{x}_i)$.

As commonly the case in spectral/*hp* methods, we will also use this nodal interpretation for the explicit treatment of the (non-linear) advection term. The term $\hat{\boldsymbol{f}}$ in Eq. (4.34) will then be computed as

$$\hat{\boldsymbol{f}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{f}, \tag{4.40}$$

where $\boldsymbol{f}$ represents the original advection term evaluated at the quadrature points, i.e. $\boldsymbol{f}[i] = f(u(\boldsymbol{x}_i))$ and $\boldsymbol{B}^\top \boldsymbol{W}$ is the discrete inner product operator, see Section 2.2.4.3. Such a collocation approach is also known as the *pseudo-spectral* method (Gottlieb & Orszag 1977).

4.3.3.1 The Helmholtz problem and the projection problem

Before we derive the new framework, we will first introduce the following two concepts which will facilitate the derivation.

The projection problem Consider the discrete solution space $\mathcal{U}^\delta(\Omega, t)$ of C^0 continuous piecewise polynomial functions that satisfy the (possibly time-dependent) Dirichlet boundary conditions. We define the projection of an arbitrary function $f(\boldsymbol{x})$, denoted as

$$u = \mathcal{P}(f, t), \tag{4.41}$$

as the L^2 projection of f onto $\mathcal{U}^\delta(\Omega)$. This projection is equivalent to solving the following minimisation problem using a traditional Galerkin finite element approach: Find $u \in \mathcal{U}^\delta(\Omega)$ such that $||u - f||_{L^2}$ is minimal. In a nodal/collocated context, this projection can be computed as:

- in case of strongly enforced Dirichlet boundary conditions

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t) \\ &= \begin{bmatrix} \boldsymbol{B}^D & \boldsymbol{B}^H \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}^D(t) \\ \left(\boldsymbol{M}^{HH}\right)^{-1} \left\{ \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W} \boldsymbol{f} - \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}^D(t) \right\} \end{bmatrix}, \end{aligned} \tag{4.42}$$

- which in the absence of strongly enforced Dirichlet boundary conditions, simplifies to

$$\boldsymbol{u} = \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t) = \boldsymbol{B} \boldsymbol{M}^{-1} \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{f}. \tag{4.43}$$

The Helmholtz problem Given an arbitrary function f, we define the Helmholtz problem as finding the Galerkin finite element solution to the (steady) elliptic

Helmholtz equation

$$u - \lambda \nabla^2 u = f, \qquad \text{in } \Omega, \tag{4.44a}$$

$$u(\boldsymbol{x}) = g_D(\boldsymbol{x}), \qquad \text{on } \partial\Omega_D, \tag{4.44b}$$

$$\frac{\partial u}{\partial n}(\boldsymbol{x}) = \boldsymbol{g}_N(\boldsymbol{x}) \cdot \boldsymbol{n}, \qquad \text{on } \partial\Omega_N. \tag{4.44c}$$

We will also denote this problem as

$$u = \mathcal{H}(f, \lambda, t). \tag{4.45}$$

Again adopting a nodal/collocated interpretation of the spectral/*hp* expansion, this problem can be evaluated as:

- in case of strongly enforced Dirichlet boundary conditions

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{\mathcal{H}}(\boldsymbol{f}, \lambda, t) \\ &= \begin{bmatrix} \boldsymbol{B}^D & \boldsymbol{B}^H \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}^D(t) \\ \left(\boldsymbol{H}^{HH}\right)^{-1} \left\{ \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W}\boldsymbol{f} + \lambda \boldsymbol{\Gamma}^H(t) - \boldsymbol{H}^{HD}\hat{\boldsymbol{u}}^D(t) \right\} \end{bmatrix}, \end{aligned} \tag{4.46}$$

- which in the absence of strongly enforced Dirichlet boundary conditions, simplifies to

$$\boldsymbol{u} = \boldsymbol{\mathcal{H}}(\boldsymbol{f}, \lambda, t) = \boldsymbol{B}\boldsymbol{H}^{-1} \left(\boldsymbol{B}^\top \boldsymbol{W}\boldsymbol{f} + \lambda \boldsymbol{\Gamma}(t) \right). \tag{4.47}$$

In the expressions above, the matrix $\boldsymbol{H}$ represents the Helmholtz matrix defined as

$$\boldsymbol{H}[i][j] = \int_\Omega \Phi_i \Phi_j + \lambda \nabla \Phi_i \cdot \nabla \Phi_j d\boldsymbol{x} \quad i, j \in \mathcal{N}. \tag{4.48}$$

Properties We will use the following properties of the operators $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{H}}$ in the subsequent sections:

- In case $\lambda = 0$, the operator $\boldsymbol{\mathcal{H}}$ reduces to the projection operator $\boldsymbol{\mathcal{P}}$, i.e.

$$\boldsymbol{\mathcal{H}}(\boldsymbol{f}, 0, t) = \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t). \tag{4.49}$$

- Composition of the operators

$$\boldsymbol{\mathcal{P}}(\boldsymbol{\mathcal{P}}(\boldsymbol{f},t_m),t_n)=\boldsymbol{\mathcal{P}}(\boldsymbol{f},t_n), \tag{4.50}$$

$$\boldsymbol{\mathcal{H}}(\boldsymbol{\mathcal{P}}(\boldsymbol{f},t_m),t_n)=\boldsymbol{\mathcal{H}}(\boldsymbol{f},t_n), \tag{4.51}$$

$$\boldsymbol{\mathcal{P}}(\boldsymbol{g}+\boldsymbol{\mathcal{P}}(\boldsymbol{f},t_m),t_n)=\boldsymbol{\mathcal{P}}(\boldsymbol{g}+\boldsymbol{f},t_n), \text{and} \tag{4.52}$$

$$\boldsymbol{\mathcal{H}}(\boldsymbol{g}+\boldsymbol{\mathcal{P}}(\boldsymbol{f},t_m),t_n)=\boldsymbol{\mathcal{H}}(\boldsymbol{g}+\boldsymbol{f},t_n). \tag{4.53}$$

Concerning the computational complexity, we would like to note that each operator involves a single global system inverse.

4.3.3.2 Derivation of the framework

According to Eq. (4.15a), the calculation of the i^{th} stage (for convenience of notation denoted as $\hat{\boldsymbol{u}}_i^H$) of an arbitrary GLM applied to Eq. (4.33) can be represented as

$$\begin{aligned}\hat{\boldsymbol{u}}_i^H =&\Delta t\sum_{j=1}^{i}a_{ij}^{\mathrm{IM}}\left[\left(\boldsymbol{M}^{HH}\right)^{-1}\left(\hat{\boldsymbol{g}}_j^H-\boldsymbol{M}^{HD}\frac{d\hat{\boldsymbol{u}}^D}{dt}\bigg|_j\right)\right]\\ &+\Delta t\sum_{j=1}^{i-1}a_{ij}^{\mathrm{EX}}\left[\left(\boldsymbol{M}^{HH}\right)^{-1}\hat{\boldsymbol{f}}_j^H\right]+\sum_{j=1}^{r}u_{ij}\hat{\boldsymbol{u}}_j^{H[n-1]},\end{aligned} \tag{4.54}$$

where for simplicity we have used the notation

$$\hat{\boldsymbol{g}}_j^H=-\begin{bmatrix}\boldsymbol{L}^{HD} & \boldsymbol{L}^{HH}\end{bmatrix}\begin{bmatrix}\hat{\boldsymbol{u}}_j^D\\ \hat{\boldsymbol{u}}_j^H\end{bmatrix}+\boldsymbol{\Gamma}_j^H. \tag{4.55}$$

For generality we will assume a GLM with an input/output vector of the form

$$\hat{\boldsymbol{u}}^{H[n]}=\begin{bmatrix}\hat{\boldsymbol{u}}_n^H\\ \hat{\boldsymbol{u}}_{n-1}^H\\ \Delta t\boldsymbol{G}_n\\ \Delta t\boldsymbol{G}_{n-1}\\ \Delta t\boldsymbol{F}_n\\ \Delta t\boldsymbol{F}_{n-1}\end{bmatrix}, \tag{4.56}$$

which applied to the advection-diffusion example under consideration, leads to

$$\hat{\boldsymbol{u}}^{H[n]} = \begin{bmatrix} \hat{\boldsymbol{u}}_n^H \\ \hat{\boldsymbol{u}}_{n-1}^H \\ \Delta t \left(\boldsymbol{M}^{HH}\right)^{-1} \left(\hat{\boldsymbol{g}}_n^H - \boldsymbol{M}^{HD} \left. \frac{d\hat{\boldsymbol{u}}^D}{dt} \right|_n \right) \\ \Delta t \left(\boldsymbol{M}^{HH}\right)^{-1} \left(\hat{\boldsymbol{g}}_{n-1}^H - \boldsymbol{M}^{HD} \left. \frac{d\hat{\boldsymbol{u}}^D}{dt} \right|_{n-1} \right) \\ \Delta t \left(\boldsymbol{M}^{HH}\right)^{-1} \hat{\boldsymbol{f}}_n^H \\ \Delta t \left(\boldsymbol{M}^{HH}\right)^{-1} \hat{\boldsymbol{f}}_{n-1}^H \end{bmatrix}. \tag{4.57}$$

In order to deal with the time-derivative of the Dirichlet boundary condition, we first would like to note that we have chosen to treat the term involving $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ implicitly in Eq. (4.54). However, this is an arbitrary choice and we could equally well have chosen to treat this term explicitly, leading to exactly the same framework. If we then acknowledge that the variable $\hat{\boldsymbol{u}}^D$ can be understood to satisfy the ODE

$$\left(\hat{\boldsymbol{u}}^D\right)' = \frac{d\hat{\boldsymbol{u}}^D}{dt}, \tag{4.58}$$

we can apply the same GLM to this ODE as the one we have used for the original ODE in terms of $\hat{\boldsymbol{u}}^H$, i.e. Eq. (4.54), to arrive at

$$\hat{\boldsymbol{u}}_i^D = \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \left. \frac{d\hat{\boldsymbol{u}}^D}{dt} \right|_j + \sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{D[n-1]}. \tag{4.59}$$

There are no explicit stage derivatives $\boldsymbol{F}_j$ appearing in the equation above (or more precisely, $\boldsymbol{F}_j = 0$) due to the fact that we also choose to treat the right-hand-side term $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ in Eq. (4.58) implicitly. As a result, the input/output vector of the GLM under consideration, see Eq. (4.56), applied to Eq. (4.58) takes the form

$$\hat{\boldsymbol{u}}^{D[n]} = \begin{bmatrix} \hat{\boldsymbol{u}}_n^D \\ \hat{\boldsymbol{u}}_{n-1}^D \\ \Delta t \left. \dfrac{d\hat{\boldsymbol{u}}^D}{dt} \right|_n \\ \Delta t \left. \dfrac{d\hat{\boldsymbol{u}}^D}{dt} \right|_{n-1} \\ 0 \\ 0 \end{bmatrix}. \tag{4.60}$$

To eliminate the Dirichlet derivative in Eq. (4.54), we substitute Eq. (4.59) into Eq. (4.54), yielding

$$\begin{aligned}\hat{\boldsymbol{u}}_i^H =& \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \left[\left(\boldsymbol{M}^{HH} \right)^{-1} \hat{\boldsymbol{g}}_j^H \right] + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \left[\left(\boldsymbol{M}^{HH} \right)^{-1} \hat{\boldsymbol{f}}_j^H \right] \\ &+ \left(\boldsymbol{M}^{HH} \right)^{-1} \boldsymbol{M}^{HD} \left[\sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{D[n-1]} - \hat{\boldsymbol{u}}_i^D \right] + \sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{H[n-1]}, \end{aligned} \tag{4.61}$$

which after rearranging and multiplication with $\boldsymbol{M}^{HH}$ leads to

$$\begin{aligned}\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_i^H + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_i^D =& \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \hat{\boldsymbol{g}}_j^H + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \hat{\boldsymbol{f}}_j^H \\ &+ \sum_{j=1}^{r} u_{ij} \left[\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_j^{H[n-1]} + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_j^{D[n-1]} \right], \end{aligned} \tag{4.62}$$

or

$$\begin{aligned}\boldsymbol{H}^{HH} \hat{\boldsymbol{u}}_i^H + \boldsymbol{H}^{HD} \hat{\boldsymbol{u}}_i^D =& \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \hat{\boldsymbol{g}}_j^H + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \hat{\boldsymbol{f}}_j^H \\ &+ \sum_{j=1}^{r} u_{ij} \left[\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_j^{H[n-1]} + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_j^{D[n-1]} \right] + a_{ii}^{\mathrm{IM}} \Delta t \boldsymbol{\Gamma}_i^H, \end{aligned} \tag{4.63}$$

where $\boldsymbol{H}$ is the Helmholtz matrix, see Eq. (4.48), with $\lambda = a_{ii}^{\mathrm{IM}} \Delta t$. This elimination of $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ appears to give rise to a modified input/output vector $\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_j^{H[n-1]} + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_j^{D[n-1]}$, which after combining Eq. (4.57) and Eq. (4.60) can be appreciated to be equal to

$$\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}^{H[n]} + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}^{D[n]} = \begin{bmatrix} \boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_n^H + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_n^D \\ \boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_{n-1}^H + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_{n-1}^D \\ \Delta t \hat{\boldsymbol{g}}_n^H \\ \Delta t \hat{\boldsymbol{g}}_{n-1}^H \\ \Delta t \hat{\boldsymbol{f}}_n^H \\ \Delta t \hat{\boldsymbol{f}}_{n-1}^H \end{bmatrix}. \tag{4.64}$$

Following a collocation approach to calculate the advection term, see Eq. (4.40), and adopting a nodal interpretation for the solution values at the time-levels n and $n-1$ according to Eq. (4.39), this input/output vector can be considered as the

inner product of an input/output vector $\boldsymbol{u}^{[n]}$ in physical space, i.e.

$$\boldsymbol{M}^{HH}\hat{\boldsymbol{u}}^{H[n]} + \boldsymbol{M}^{HD}\hat{\boldsymbol{u}}^{D[n]} = \left(\boldsymbol{B}^{H}\right)^{\top}\boldsymbol{W}\boldsymbol{u}^{[n]} = \left(\boldsymbol{B}^{H}\right)^{\top}\boldsymbol{W}\begin{bmatrix} \boldsymbol{u}_n \\ \boldsymbol{u}_{n-1} \\ \Delta t\boldsymbol{g}_n \\ \Delta t\boldsymbol{g}_{n-1} \\ \Delta t\boldsymbol{f}_n \\ \Delta t\boldsymbol{f}_{n-1} \end{bmatrix}, \tag{4.65}$$

where we have made use of the relationship $\boldsymbol{M}^{HD} = \left(\boldsymbol{B}^{H}\right)^{\top}\boldsymbol{W}\boldsymbol{B}^{D}$. In addition, we have also adopted a nodal interpretation $\boldsymbol{g}_n$ of the implicit stage value $\hat{\boldsymbol{g}}_n^H$ which will be further discussed in the next section. Making use of this formulation in physical space, Eq. (4.63) can be written as

$$\begin{aligned}\boldsymbol{H}^{HH}\hat{\boldsymbol{u}}_i^H + \boldsymbol{H}^{HD}\hat{\boldsymbol{u}}_i^D = \left(\boldsymbol{B}^{H}\right)^{\top}\boldsymbol{W}\left\{\Delta t\sum_{j=1}^{i-1}a_{ij}^{\mathrm{IM}}\boldsymbol{g}_j + \Delta t\sum_{j=1}^{i-1}a_{ij}^{\mathrm{EX}}\boldsymbol{f}_j + \sum_{j=1}^{r}u_{ij}\boldsymbol{u}_j\right\} \\ + a_{ii}^{\mathrm{IM}}\Delta t\boldsymbol{\Gamma}_i^H.\end{aligned} \tag{4.66}$$

Finally also adopting a nodal interpretation $\boldsymbol{u}_i$ for the solution at stage i, the calculation of the i^{th} stage value can be recognised as

$$\boldsymbol{u}_i = \boldsymbol{\mathcal{H}}\left(\Delta t\sum_{j=1}^{i-1}a_{ij}^{\mathrm{IM}}\boldsymbol{g}_j + \Delta t\sum_{j=1}^{i-1}a_{ij}^{\mathrm{EX}}\boldsymbol{f}_j + \sum_{j=1}^{r}u_{ij}\boldsymbol{u}_j, a_{ii}^{\mathrm{IM}}\Delta t, t_n\right). \tag{4.67}$$

This formulation allows for a well defined procedure to advance the solution in time as the calculation of the stage values only involves solving the associated Helmholtz problem. Note that for a pure explicit method, the Helmholtz problem reduces to the L^2 projection. This solution procedure is very attractive in particularly for the following three reasons:

- uniform treatment of Dirichlet boundary conditions (i.e. the Dirichlet boundary conditions only come into play when enforcing them while solving the global matrix system),
- only one global matrix inverse is required per stage, and
- it is sufficiently generic to be extended to the entire range of GLMs (see next section).

Note that the derivation of this framework was founded on the following two steps:

- adopting a consistent-order discretisation of the Dirichlet derivative consistent to the discretisation of the original ODE, and
- formulating the GLM algorithm in physical space.

4.3.3.3 Algorithm

A PDE time-stepping algorithm that time-integrates the advection-diffusion equation (4.26) from time-level $n-1$ to n can then be formulated as:

input : the vector $\boldsymbol{u}^{[n-1]}$

output: the vector $\boldsymbol{u}^{[n]}$

`// Calculate stage values` $\boldsymbol{U}_i$ `and the stage derivatives` $\boldsymbol{F}_i$ `and` $\boldsymbol{G}_i$

for $i = 1$ **to** s **do**

`// calculate the temporary variable` $\boldsymbol{x}_i$

(A2.1) $\quad \boldsymbol{x}_i = \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{u}_j^{[n-1]}$

`// calculate the stage value` $\boldsymbol{U}_i$

(A2.2) $\quad \boldsymbol{U}_i = \boldsymbol{\mathcal{H}}\left(\boldsymbol{x}_i, a_{ii}^{\mathrm{IM}} \Delta t, t_i\right)$

`// calculate the explicit stage derivative` $\boldsymbol{F}_i$

(A2.3) $\quad \boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{U}_i)$

`// calculate the implicit stage derivative` $\boldsymbol{G}_i$

(A2.4) $\quad \boldsymbol{G}_i = \frac{1}{a_{ii}^{\mathrm{IM}} \Delta t} \left(\boldsymbol{U}_i - \boldsymbol{x}_i\right)$

end

`// Calculate the output vector` $\boldsymbol{u}^{[n]}$

if *last stage equals new solution* **then**

(A2.5) $\quad \boldsymbol{u}_n = \boldsymbol{U}_s$

else

(A2.6)

$$\boldsymbol{u}_1^{[n]} = \boldsymbol{u}_1^{[n]} = \boldsymbol{\mathcal{P}}\left(\Delta t \sum_{j=1}^{s} b_{1j}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} b_{1j}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{1j} \boldsymbol{u}_j^{[n-1]}, t_n\right)$$

end

for $i = 2$ **to** r **do**

(A2.7) $\quad \boldsymbol{u}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{u}_j^{[n-1]}$

end

Algorithm 2: A GLM-based PDE time-stepping algorithm.

The only subtlety that arises in comparison with the derivation as explained in the previous section is due to the term $\boldsymbol{G}_i$. To formally fit into the framework, a proper definition of $\boldsymbol{G}_i$, denoted as $\boldsymbol{g}_i$ in the previous section, should be

$$\boldsymbol{g}_i = \boldsymbol{B}^H \left(\boldsymbol{M}^{HH}\right)^{-1} \hat{\boldsymbol{g}}_i^H, \tag{4.68}$$

which can be appreciated when comparing Eq. (4.64) with Eq. (4.65). By re-combining the terms in the Helmholtz problem associated to step (A2.2) it can be demonstrated that this term can be evaluated as

$$\boldsymbol{g}_i = \boldsymbol{\mathcal{P}}\left(\frac{\boldsymbol{U}_i - \boldsymbol{x}_i}{a_{ii}^{\mathrm{IM}}\Delta t}, t_i\right), \tag{4.69}$$

which corresponds to the L^2 projection of our definition of $\boldsymbol{G}_i$, i.e. $\boldsymbol{g}_i = \boldsymbol{\mathcal{P}}(\boldsymbol{G}_i, t)$ However, do due the properties (4.52-4.53) it can be appreciated that using $\boldsymbol{G}_i$ in steps (A2.1), (A2.6) and (A2.7) is equivalent to the use of $\boldsymbol{g}_i$, thereby keeping the number of global system inverses per stage to one.

Comparing this PDE time-stepping algorithm with the original ODE algorithm (Algorithm 1), one can identify the following differences:

- While step (A1.2) of the original algorithm essentially is a pure algebraic problem (that is, for schemes with an implicit component), step (A2.2) also as an broader analytical interpretation in the sense that it is the solution of the elliptic Helmholtz partial differential equation.

- It has been indicated before that for purely explicit time-stepping methods (i.e. $a_{II}^{\mathrm{IM}} = 0$), the evaluation of step (A1.1) actually vanishes as it is reduced to $\boldsymbol{Y}_i = \boldsymbol{x}_i$. However, in the new algorithm, step (A2.2) now also requires a global system inverse, as the Helmholtz problem is reduced to the projection problem for explicit schemes.

- In addition, step (A2.6) of the new algorithm also requires a L^2 projection. This is necessary to ensure that the solution is in the solution space, as the right-hand-side cannot be guaranteed to be in this space. Note that this requires an additional global system inverse.

- In order to possibly eliminate the cost associated to this additional global system inverse in step (A2.6), we have included an optimisation check in step (A2.5). In case the last row of coefficient matrices A and U is equal to the first row of respectively the matrices B and V, the last stage $\boldsymbol{U}_s$ is equal to the new solution $\boldsymbol{u}_n$ and the (expensive) evaluation of step (A2.6) can be

omitted. Fortunately, all multi-step schemes – and many more methods as can be seen from the GLM tableau's (4.10,4.12,4.13,4.20) – can be formulated to satisfy this condition. As a result, evaluating any linear multi-step method for a single time-step based upon the proposed algorithm only requires a single global system inverse.

In order to evaluate this PDE time-stepping algorithm for an arbitrary general linear method, the user should supply the framework with the following three *external* routines:

- a routine that evaluates the advection term $f(u,t)$ according to the spatial discretisation scheme at the quadrature/collocation points (in order to evaluate step (A.2.3)),
- a routine which solves the projection problem of Section 4.3.3.1 (in order to evaluate step (A.2.2) in case $a_{ii}^{\mathrm{IM}} = 0$ and step (A.2.6)), and
- a routine that solves the Helmholtz equation of Section 4.3.3.1 (in order to evaluate step (A.2.2) in case $a_{ii}^{\mathrm{IM}} \neq 0$).

Recall that all three routines should be defined in *physical space*, i.e. both input and output arrays correspond to functions evaluated at the quadrature/collocation points.

Remark 1 *Although the framework has been derived by means of the advection-diffusion equation, it is also applicable for other PDEs as well (see also Section 4.4). The advection term $f(u)$ could in principle also represent a possible reaction term, while the diffusion term $\nabla^2 u$ could be replaced by a more general term $g(u)$ to be treated implicitly. In the latter case, the user should supply the framework with a routine that solves the PDE $u - \lambda g(u) = f$ rather than the Helmholtz equation.*

Remark 2 *Because the algorithm is formulated in physical space, the proposed framework is applicable independently of the choice of spatial discretisation technique. E.g. it can be verified that the algorithm is also valid in case of a Discontinuous Galerkin discretisation (see also Section 4.4).*

4.3.3.4 Object-oriented implementation

The class structure presented in Section 4.2.2 that implements the ODE framework can be slightly modified in order to accommodate this PDE time-stepping framework. An overview of the required classes is shown in Figure 4.2. The underlying idea remains identical and only the class `TimeIntegrationOperators` should be altered to take into account the projection operator. Therefore, this class should now be equipped with an additional function pointer `m_project` that points to the implementation of the L^2 projection operator, i.e. Eq. (4.42). In order to set and evaluate this function pointer, the class now also contains the methods `DefineProject` and `DoProject`.

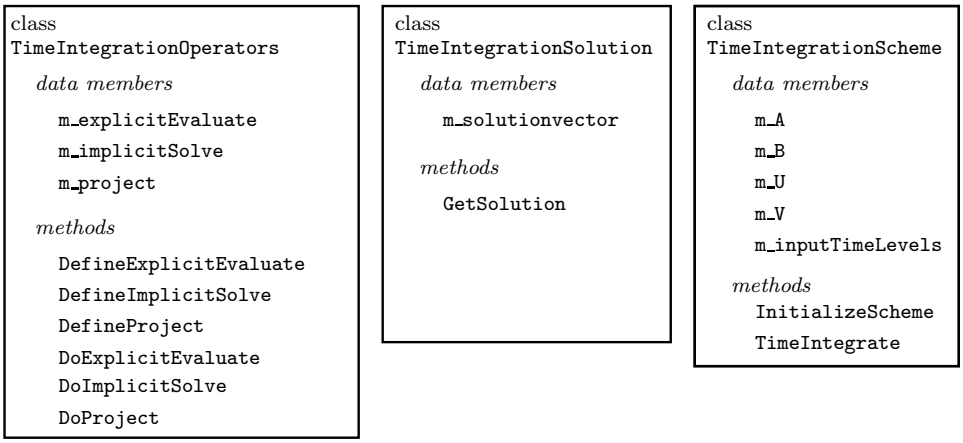


Figure 4.2: Overview of the classes in the implementation of the generic PDE time-stepping framework.

4.4 Computational Examples

In this section we present examples demonstrating the capabilities of the PDE time-stepping framework presented in the previous section. First we verify the algorithm by considering an advection-diffusion problem and then we apply the framework to two fluid problems: standing waves in shallow water and vortex shedding around a cylinder. In all our examples we use the spectral/*hp* element method for the spatial discretisation. If not mentioned otherwise, we use the standard C^0-continuous Galerkin version in the following.

4.4.1 Linear advection-diffusion equation

As a first example, we investigate the popular linear advection-diffusion problem of a Gaussian hill convected with a velocity $\boldsymbol{V}$ while spreading isotropically with a diffusivity ν (Donea, Giuliani, Laval, & Quartapelle 1984; Noye & Tan 1989). The analytical solution is given by

$$u(\boldsymbol{x}, t) = \frac{1}{4t+1} \exp\left(-\left(\frac{x - x_0 - V_x t}{\nu(4t+1)}\right) - \left(\frac{y - y_0 - V_y t}{\nu(4t+1)}\right)\right). \tag{4.70}$$

The problem is discretised in space on an unstructured triangular mesh of 84 elements and using a 12^{th}-order spectral/hp element expansion. The Gaussian hill is initially located a $\boldsymbol{x}_0 = [0.5\,, 0.5]$ and is convected with a velocity $\boldsymbol{V} = [1\,, 1]$ for one time unit and the diffusivity is set to $\nu = 0.05$. Time-dependent Dirichlet boundary conditions given by the analytical solution are imposed on the domain boundaries.

We have applied the time-stepping framework on this equation using two different time-stepping schemes: the 2^{nd}- and the 3^{rd}-order IMEX-DIRK scheme as respectively presented in (Ascher, Ruuth, & Wetton 1995; Ascher, Ruuth, & Spiteri 1997), see also Eq. (4.22). Therefore, we have supplied the framework with the three necessary routines as explained in Section 4.3.3.3, i.e. a function that evaluates the linear advection term, a projection operator and a Helmholtz solver. In order to verify that the framework integrates the PDE correctly, we have checked the order of convergence in function of the time-step Δt for both the IMEX schemes. Considering Fig. 4.3, it can be observed that the L^2 error converges according to the expected rate when using the framework.

4.4.2 Shallow water equations

The shallow water equations (SWE) are frequently used for simulating flows in shallow coastal regions and rivers, for example storm surges, tsunamis and river flooding. The SWE can be formulated as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{u}) = 0. \tag{4.71}$$

For an appropriate definition of the flux term $\boldsymbol{F}$ the reader is referred to (Eskilsson & Sherwin 2004). Due to their hyperbolic nature, explicit methods are typically

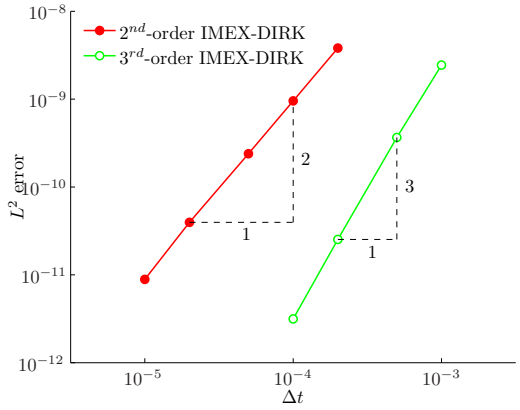


Figure 4.3: Error convergence in function of Δt for the 2nd- and the 3rd-order IMEX-DIRK scheme when using the PDE time-stepping framework of Section 4.3.3.

employed for time-stepping the SWE. As a result, the user should only provide a proper implementation of the term $\nabla \cdot \boldsymbol{F}(\boldsymbol{u})$ together with a projection method in order to use the framework.

The objective of this example is twofold:

- We would like to demonstrate that the framework can be applied with different spatial discretisation techniques. Therefore, we solve the SWE using both the discontinuous Galerkin (DG) method and the C^0 continuous Galerkin (CG) method. Note that in the former case, the user-supplied function that evaluates the flux term should include the numerical flux term typical to the DG method.

- We also want to compare the computational efficiency of the framework. Therefore, we will compare the run-time of solving the SWE using the framework with a specialised implementation of the chosen time-stepping schemes.

Within the DG community the third-order three-stage Strong-Stability-Preserving (SSP) RK scheme proposed by (Shu 1987) has emerged as the standard time-stepping scheme for the SWE. We will therefore select this time-stepping scheme together with the third-order four-stage SSP-RK scheme (Ruuth & Spiteri 2004) for the SWE test-case.

We compute the simple case of two super-positioned standing linear waves (one wave aligned in the x_1 direction and wave aligned in the x_2 direction) in a basin

of constant depth with slip conditions at the wall boundaries. For the DG method the boundary conditions are implemented weakly through the use of the numerical flux using standard mirroring technique, while for the CG method we apply $u = 0$, $\partial v/\partial n = 0$, $\partial \zeta/\partial n = 0$ at the north/south boundaries and $\partial u/\partial n = 0$, $v = 0$, $\partial \zeta/\partial n = 0$ at the east/west boundaries, respectively.

We are using a 3$^{\text{rd}}$-order spectral/*hp* element method on a mesh of 16 quadrilateral element and we solve the problem for $t \in [0\,, 100\,T]$ wave where T denotes the wave period. Table 4.1 present the required time step, the measured run-time and memory usage (based upon the heap profiler Massif of Valgrind's tool suite) for obtaining an error less than 1×10^{-4}. First of all, it can be concluded that the framework has been successfully applied for both the spatial discretisation techniques. We can also see from Table 4.1 that the run-time overhead of using the framework is only a couple of percent when compared to the scheme-specific implementations while the memory usage is equal. As a result, we can conclude that next to the high-level of generality, the framework also is competitive with scheme-specific implementations in terms of performance. Finally, even though not the topic of this comparison, one may observe that the superior dispersion properties of the DG method (see e.g. (Bernard, Deleersnijder, Legat, & Remacle 2008)) compared to the CG method do not become apparent, witness the fact that a similar discretisation yields a similar error. This may be explained by the fact that due to the solution's low wave-number, both Galerkin techniques resolve the solution in space up to high accuracy and the error can mainly be attributed to the temporal discretisation.

4.4.3 Incompressible Navier-Stokes equation

As an illustrative example of the use of the time-integration framework to solve more complex fluid dynamics problems, we consider a DNS simulation of the two-dimensional flow past a circular cylinder in a free stream. The incompressible Navier-Stokes (NS) equations are solved using a spectral/*hp* element discretisation in space combined with the high-order stiffly stable splitting scheme of (Karniadakis, Israeli, & Orszag 1991) for the discretisation in time. In this splitting scheme, each time step

Table 4.1: The SWE for standing waves. Computational results for obtaining an L^2 error less than 1×10^{-4}.

		DG method		CG method	
		SSP-RK(3.3)	SSP-RK(3.4)	SSP-RK(3.3)	SSP-RK(3.4)
	Δt	$T/88$	$T/70$	$T/88$	$T/70$
	L^2 error	9.76E-05	9.70E-05	9.79E-05	9.73E-05
GLM	Run-time	20.9 s	22.0 s	15.6 s	16.6 s
framework	Storage	10.1 MB	10.1 MB	5.4 MB	5.4 MB
Specialised	Run-time	20.4 s	21.4 s	15.4 s	16.3 s
implementation	Storage	10.1 MB	10.1 MB	5.4 MB	5.4 MB

is subdivided into three substeps, and the solution of the discretised Navier-Stokes equation is advanced from time-step $n-1$ to time-step n as follows:

$$\frac{\breve{\boldsymbol{u}} - \sum_{q=1}^{J_i} \alpha_q \boldsymbol{u}^{n-q}}{\Delta t} = -\sum_{q=1}^{J_e-1} \beta_q \left[(\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} \right]^{n-q}, \tag{4.72a}$$

$$\nabla^2 p^n = \nabla \cdot \left(\frac{\breve{\boldsymbol{u}}}{\Delta t} \right), \tag{4.72b}$$

$$\frac{\gamma_0 \boldsymbol{u}^n - \breve{\boldsymbol{u}}}{\Delta t} = \nu \nabla^2 \boldsymbol{u}^n - \nabla p^n. \tag{4.72c}$$

The splitting scheme decouples the velocity field $\boldsymbol{u}$ from the pressure p, leading to an explicit treatment of the advection term and an implicit treatment of the pressure and the diffusion term. J_i is the integration order for the implicit terms and J_e is the integration order for the explicit terms. The values of the coefficients γ_0 , α_q and β_q of this multi-step IMEX scheme are given in Table 4.2 for different orders. In order to use the PDE time-stepping framework of Section 4.3, we first formulate the stiffly stable scheme as a general linear method. For the second-order variant

Table 4.2: Stiffly stable splitting scheme coefficients.

	1^{st}-order	2^{nd}-order	3^{rd}-order
γ_0	1	3/2	11/6
α_0	1	2	3
α_1	0	$-1/2$	$-3/2$
α_2	0	0	1/3
β_0	1	2	3
β_1	0	-1	-3
β_2	0	0	1

for example, this yields

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cccc} \frac{2}{3} & 0 & \frac{4}{3} & -\frac{1}{3} & \frac{4}{3} & -\frac{2}{3} \\ \hline \frac{2}{3} & 0 & \frac{4}{3} & -\frac{1}{3} & \frac{4}{3} & -\frac{2}{3} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right], \tag{4.73}$$

with

$$\boldsymbol{y}^{[n]} = \begin{bmatrix} \boldsymbol{y}_n \\ \boldsymbol{y}_{n-1} \\ \Delta t \boldsymbol{F}_n \\ \Delta t \boldsymbol{F}_{n-1} \end{bmatrix}. \tag{4.74}$$

where the values in the first two rows have been scaled with γ_0 compared to the values in Table 4.2. Furthermore, we need to properly define the external functions needed for the time-stepping framework:

- For the explicit term, this should be a function $\boldsymbol{f}$ that evaluates the advection term, i.e.

$$\boldsymbol{f}(\boldsymbol{u}) = -\left(\boldsymbol{u} \cdot \nabla\right) \boldsymbol{u}. \tag{4.75}$$

  Because we will follow a pseudo-spectral approach for the advection term, this term should simply be evaluated at the quadrature/collocation points.

- The projection operator to be provided to the system is identical to the one defined in Section 4.3.3.1.

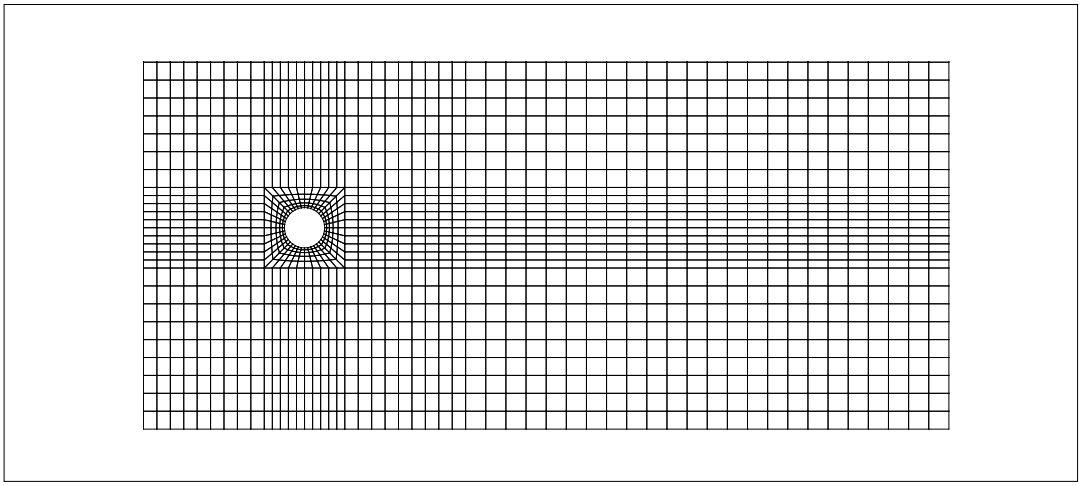
- For the implicit part of the scheme, a routine that solves the following problem is required. Given an arbitrary function $\boldsymbol{f}$, a scalar λ and a time-level t, find the velocity field u such that

$$\nabla^2 p = \nabla \cdot (\frac{\boldsymbol{f}}{\lambda}), \tag{4.76a}$$

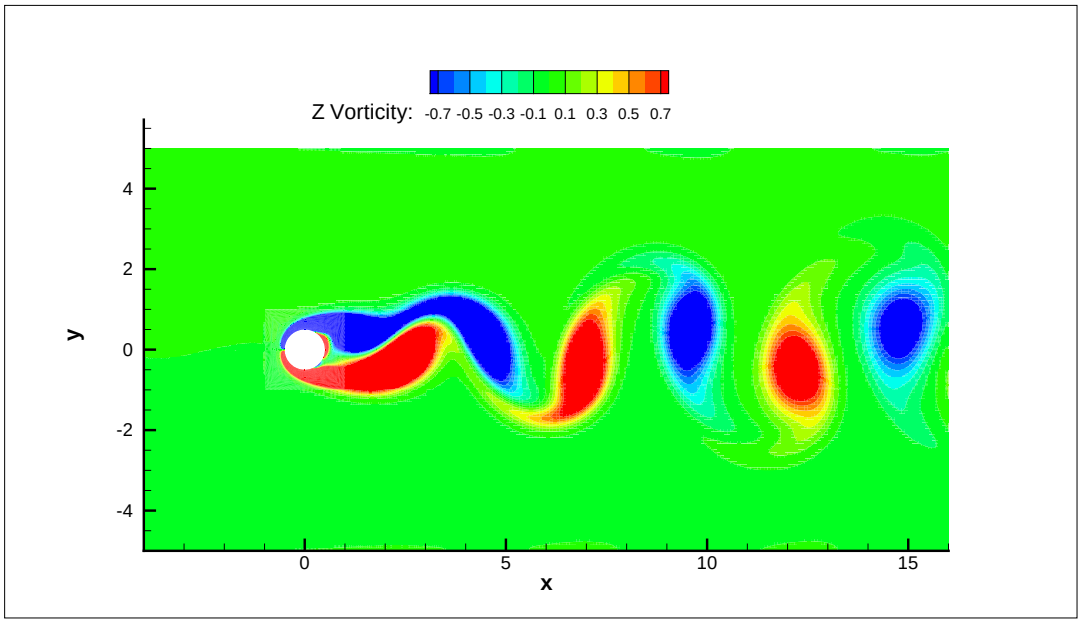
$$\boldsymbol{u} - \nu\lambda\nabla^2\boldsymbol{u} = \boldsymbol{f} - \lambda\nabla p, \tag{4.76b}$$

  and subject to the appropriate boundary conditions. It can be observed that this problem involves the consecutive solution of three elliptic problems: a Poisson problem and two (in 2D) scalar Helmholtz problems. This routine can also be used to solve the *unsteady Stokes* equations.

Figure 4.4(b) shows the vorticity field of a flow past a cylinder with a $Re = 100$ after solving the NS equations with the framework presented in Section 4.3. The solution, which highlights the vortex shedding, has been obtained using the 3^{rd}-order stiffly stable splitting scheme with $\Delta t = 0.0001$ and 7^{th}-order spectral/hp expansion on a mesh of 1500 quadrilaterals as shown in 4.4(a). The cylinder has a diameter $D = 0.4$ and the domain is defined by a rectangle $\boldsymbol{x} \in [-4\,, 16] \times [-5\,, 5]$ as shown in Figure 4.4. Boundary conditions are of Dirichlet type at the inflow, where a constant velocity in x-direction is imposed ($u = 1$ and $v = 0$) and of Neumann type (homogeneous) at the outflow and on the upper and lower domain limits. Similar results have been obtained by different authors with other techniques, e.g. (Song & Yuan 1990; Souza Carmo 2009).



(a) Computational mesh



(b) Vorticity field

Figure 4.4: Flow past a circular cylinder at $Re = 100$.

Chapter 5

Evaluation of Spectral/*hp* Element Operators

We have seen in Section 2.1, that when following a standard Galerkin procedure, the weak form of a partial differential equations often comprise terms of the form

$$a(v, u), \tag{5.1}$$

where $a(\cdot, \cdot)$ is a bi-linear form and u and v are functions respectively belonging to a suitably chosen trial space $\mathcal{U}$ and test space $\mathcal{V}$. After discretising both u and v using (2.54), this yields expressions of the form

$$\hat{y}_i^g := \sum_{j \in \mathcal{N}^g} a(\Phi_i, \Phi_j) \hat{u}_j^g, \tag{5.2}$$

which should typically be evaluated for all $i \in \mathcal{N}^g$. Using the matrix associated to this bi-linear form, this finite element operation can be written as

$$\hat{\boldsymbol{y}}_g = \boldsymbol{A} \hat{\boldsymbol{u}}_g, \tag{5.3}$$

where $\boldsymbol{A}[i][j] = a(\Phi_i, \Phi_j)$.

The goal of this chapter is to analyse how the operators of type (5.2) or (5.3) can be efficiently *evaluated* for both low- and high-order expansions. As a result, the focus is not on the matrix $\boldsymbol{A}$ itself, but merely on the action of $\boldsymbol{A}$. We seek for the most efficient way of mapping the vector $\hat{\boldsymbol{u}}_g$ to the vector $\hat{\boldsymbol{y}}_g$, without necessarily building $\boldsymbol{A}$ explicitly.

The action of finite element operators is required in various scenarios. Examples include:

- iterative solution techniques, which are founded on the forward operation of the matrix to be inverted,
- explicit time-stepping methods, which require an operator evaluation to compute the right-hand-side term of the semi-discrete system (see also Chapter 4), and
- the strong enforcement of non-homogeneous (essential) Dirichlet boundary conditions (see also Chapter 4 and Section 7.1).

5.1 Evaluation strategies

In this section, we discuss three different strategies to evaluate expressions of the form (5.2). The first two strategies – referred to as the *sum-factorisation technique* and the *local matrix approach* – have in common that they both exploit the elemental decomposition of the spectral/*hp* element method. The contribution of each element is computed separately, where after the different contributions are assembled together using the direct stiffness summation technique as explained in Section 2.3. Both these approaches then evaluate (5.2) in the following three steps:

1. global-to-local mapping: $\hat{\boldsymbol{u}}_l = \mathcal{A}\hat{\boldsymbol{u}}_g$, (5.4)
2. elemental evaluation: $\hat{y}_m^e = \sum_{n\in\mathcal{N}} a_e(\phi_m^e, \phi_n^e)\hat{u}_n^e \quad \forall m \in \mathcal{N}, e \in \mathcal{E}$, (5.5)
3. global assembly: $\hat{\boldsymbol{y}}_g = \mathcal{A}^\top \hat{\boldsymbol{y}}_l$. (5.6)

The question then becomes: how can the elemental form (5.5) be evaluated efficiently?

The third strategy - using *global matrices* - differs from both these approaches in the sense that it is based on the global interpretation of the spectral/*hp* element method, rather than on its elemental decomposition.

A graphical representation of the different strategies is shown in Figure 5.1. The existence of different evaluation strategies has been acknowledged before in

(Bagheri, Scott, & Zhang 1994; Kirby, Knepley, & Scott 2004). In the present work, we will elaborate on a detailed cost analysis both from a theoretical (Section 5.2) as computational (Section 5.3) point of view, and we particularly emphasise on the difference in results depending on the expansion order.

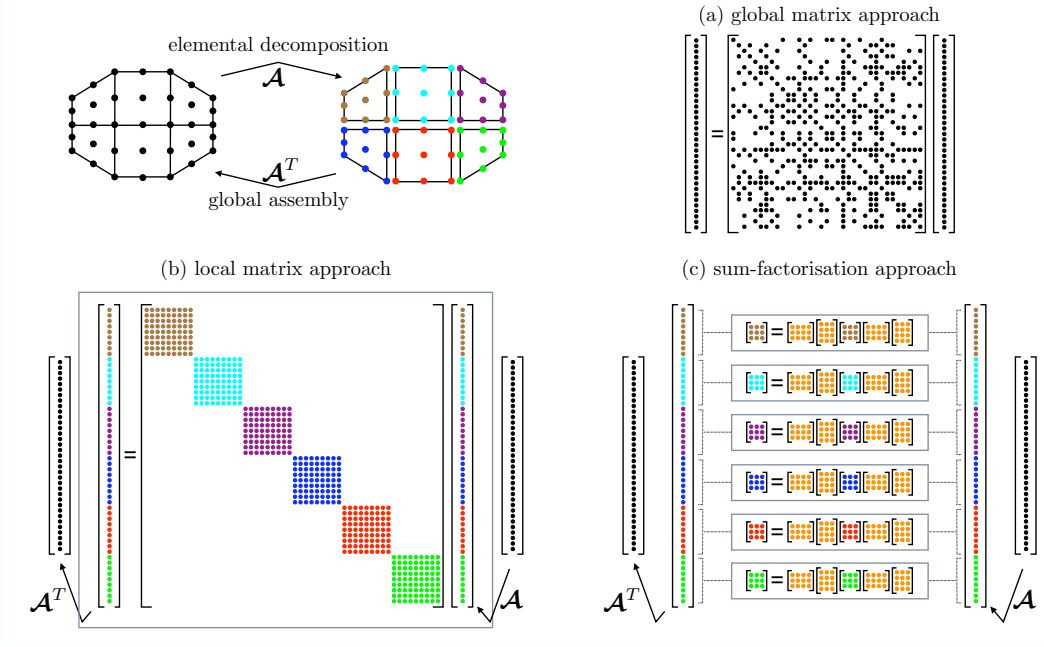


Figure 5.1: A graphical representation of the different implementation strategies for evaluating a spectral/*hp* operator.

5.1.1 The sum-factorisation approach

To introduce the sum-factorisation technique, consider the example of the mass matrix operator. The elemental bi-linear form appearing in (5.5) is then defined as

$$a_e(\phi_m, \phi_n) = \int_{\Omega_e} \phi_m(\boldsymbol{x})\phi_n(\boldsymbol{x})d\boldsymbol{x}. \tag{5.7}$$

Making use of the coordinate transformation (2.12), this can be expressed as an integral over the reference domain Ω_{st}

$$a_e(\phi_m, \phi_n) = \int_{\Omega_{st}} \phi_m(\boldsymbol{\xi})\phi_n(\boldsymbol{\xi})|J(\boldsymbol{\xi})|d\boldsymbol{\xi}, \tag{5.8}$$

where J is the Jacobian of the transformation.

The sum-factorisation technique is based upon two concepts: *numerical quadrature* and the use of tensorial basis functions.

5.1.1.1 Evaluation using numerical quadrature

Assume a quadrature rule of order Q to evaluate the integral

$$\int_{\Omega_{st}} f(\boldsymbol{\xi}) d\boldsymbol{\xi} = \sum_{r \in \mathcal{Q}} f(\boldsymbol{\xi}_r) \omega_r. \tag{5.9}$$

Inserting (5.9) and (5.8) into (5.5) leads to the following expression to be evaluated

$$\hat{y}_m = \sum_{n \in \mathcal{N}} \sum_{r \in \mathcal{Q}} \omega_r \phi_m(\boldsymbol{\xi}_r) \phi_n(\boldsymbol{\xi}_r) |J(\boldsymbol{\xi}_r)| \hat{u}_n, \quad \forall m \in \mathcal{N}, \tag{5.10}$$

where for clarity, we have omitted the index e. After rearranging, this yields

$$\hat{y}_m = \sum_{r \in \mathcal{Q}} \phi_m(\boldsymbol{\xi}_r) \omega_r |J(\boldsymbol{\xi}_r)| \left\{ \sum_{n \in \mathcal{N}} \phi_n(\boldsymbol{\xi}_r) \hat{u}_n \right\}, \quad \forall m \in \mathcal{N}. \tag{5.11}$$

In matrix notation this simplifies to

$$\hat{\boldsymbol{y}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{B} \hat{\boldsymbol{u}}, \tag{5.12}$$

where $\boldsymbol{B}[i][j] = \phi_j(\boldsymbol{\xi}_i)$ is the discrete representation of the expansion basis and $\boldsymbol{W}$ is the diagonal matrix with entries $\boldsymbol{W}[i][i] = \omega_i |J(\boldsymbol{\xi}_i)|$ as defined in Section 2.2.4. It can be appreciated that the mass matrix operator can now be evaluated in two separate steps: a *backward transformation* $\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}$ which evaluates the spectral/*hp* expansion at the quadrature points, and the *inner product operator* $\hat{\boldsymbol{y}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{u}$ which subsequently takes the inner product of the function u with respect to all elemental expansion modes.

Remark 3 *Despite the typical formulation (5.12) in matrix notation, the numerical quadrature approach can be classified as a* matrix-free *approach. It is matrix-free in the sense that evaluation of the operator is not dependent on the construction of the matrix associated to the bi-linear form.*

5.1.1.2 Sum-factorisation for quadrilateral expansions

If both the spectral/*hp* expansion basis and the numerical quadrature rule exhibit a tensor-product structure, the backward transformation as well as the inner product

can be factorised using the sum-factorisation technique. Consider the backward transformation $\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}$ for a quadrilateral element, defined as

$$u(\boldsymbol{\xi}_r) = \sum_{n \in \mathcal{N}} \phi_n(\boldsymbol{\xi}_r)\hat{u}_n \quad \forall r \in \mathcal{Q}. \tag{5.13}$$

Acknowledging the tensorial nature of both the expansion and quadrature rule, this yields

$$\begin{aligned} u(\xi_{1s}, \xi_{2t}) &= \sum_{p=0}^{P}\sum_{q=0}^{P} \psi_p^a(\xi_{1s})\psi_q^a(\xi_{2t})\hat{u}_{pq} \\ &= \sum_{q=0}^{P} \psi_q^a(\xi_{2t}) \left\{ \sum_{p=0}^{P} \psi_p^a(\xi_{1s})\hat{u}_{pq} \right\}, \quad \forall s,t \in \{0,\ldots,Q-1\}, \end{aligned} \tag{5.14}$$

where we have *factored* the term $\psi_q^a(\xi_{2t})$ out of the second summation. In matrix notation, this is equivalent to:

$$\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}} = \left(\boldsymbol{B}_2^a \otimes \boldsymbol{B}_1^a\right)\hat{\boldsymbol{u}} = \left(\boldsymbol{B}_2^a \otimes \boldsymbol{I}_Q\right)\left\{\left(\boldsymbol{I}_{P+1} \otimes \boldsymbol{B}_1^a\right)\hat{\boldsymbol{u}}\right\}, \tag{5.15}$$

where $\boldsymbol{B}_m^a[i][j] = \psi_j^a(\xi_i)$ represents the one-dimensional basis in direction m, $\boldsymbol{I}_Q$ is the identity matrix of size $Q \times Q$ and $\otimes$ is the Kronecker product. Note that the relation $\boldsymbol{B} = \boldsymbol{B}_2^a \otimes \boldsymbol{B}_1^a$ reflects the tensorial structure of the expansion. Consequently, the backward transformation can then be evaluated in two separate steps:
A first step to compute the temporary variable $\boldsymbol{v} = \left(\boldsymbol{I}_{P+1} \otimes \boldsymbol{B}^a\right)\hat{\boldsymbol{u}}$:

$$v_q(\xi_{2s}) = \sum_{p=0}^{P} \psi_p^a(\xi_{1s})\hat{u}_{pq} \quad \forall q,s \quad , \tag{5.16}$$

followed by the evaluation of $\boldsymbol{u} = \left(\boldsymbol{B}^a \otimes \boldsymbol{I}_Q\right)\boldsymbol{v}$:

$$u(\xi_{1s}, \xi_{2t}) = \sum_{q=0}^{P} \psi_q^a(\xi_{2t})v_q(\xi_{2s}) \quad \forall s,t \quad . \tag{5.17}$$

Furthermore, if we respectively consider $\hat{\boldsymbol{u}}$ and $\boldsymbol{u}$ as the column-major vectorisation of the matrices $\hat{\boldsymbol{U}}$ and $\boldsymbol{U}$, the notation can be further simplified to (Deville, Fischer, & Mund 2002)

$$\boldsymbol{U} = \boldsymbol{B}_1^a \hat{\boldsymbol{U}} \boldsymbol{B}_2^{a\top}. \tag{5.18}$$

As a result, both sub-steps (5.16) and (5.17) can effectively be evaluated as the matrix-matrix multiplications $\boldsymbol{V} = \boldsymbol{B}_1^a\hat{\boldsymbol{U}}$ and $\boldsymbol{U} = \hat{\boldsymbol{V}}\boldsymbol{B}_2^{a\top}$ respectively. However,

note that this does require a lexicographical ordering of the two-dimensional expansion modes $\hat{u}_{n(p,q)}$ with the p index running fastest as indicated in Section 2.2.4. Also the vector $u_{r(s,t)}$ should follow a similar ordering along the first coordinate direction.

Analogously, the inner product operator $\hat{\boldsymbol{y}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{u}$ can be factorised as

$$\widehat{\boldsymbol{Y}} = \boldsymbol{B}_1^{a\top} w(\boldsymbol{U}) \boldsymbol{B}_2^a, \tag{5.19}$$

where the function w multiplies each entry $\boldsymbol{U}[i][j] = u(\xi_{1i}, \xi_{2j})$ with the corresponding quadrature metric $\omega_{i,j}|J(\xi_{1i}, \xi_{2j})|$. Consequently we can conclude that when adopting the sum-factorisation approach, the mass matrix operator can be evaluated as:

$$\widehat{\boldsymbol{Y}} = \boldsymbol{B}^{a\top} w \left(\boldsymbol{B}^a \hat{\boldsymbol{U}} \boldsymbol{B}^{a\top} \right) \boldsymbol{B}^a. \tag{5.20}$$

This notation helps to appreciate that from an implementational point of view, this corresponds to a series of matrix-matrix products.

Remark 4 *The use of numerical quadrature is not a necessary prerequisite for the application of the sum-factorisation technique. The evaluation of the elemental mass matrix operator can also be factorised as*

$$\hat{y}_{m(p',q')} = |J| \sum_{p=0}^{P} \int_{-1}^{1} \psi_p^a(\xi_1) \psi_{p'}^a(\xi_1) d\xi_1 \left\{ \sum_{q=0}^{P} \int_{-1}^{1} \psi_q^a(\xi_2) \psi_{q'}^a(\xi_2) d\xi_2 \right\} \hat{u}_n \quad \forall p', q', \tag{5.21}$$

or equivalently,

$$\hat{\boldsymbol{y}} = |J| \left(\boldsymbol{M^a} \otimes \boldsymbol{M^a} \right) \hat{\boldsymbol{u}}, \tag{5.22}$$

where $\boldsymbol{M^a}$ is the mass matrix associated with the one-dimensional reference element. However, this approach implies that $|J|$ is a constant, which cannot be generally assumed. That is why in this work, the sum-factorisation technique will always be combined with the use of numerical quadrature.

5.1.1.3 Sum-factorisation for triangular expansions

For triangular elements, the use of a *generalised* tensor product expansion, makes the sum-factorisation technique more complicated. While in Eq. (5.14), both the terms

could be equally well being factorised out of the inner summation the backward transformation for triangles can only be factored as:

$$u(\xi_{1s},\xi_{2t}) = \sum_{p=0}^{P}\sum_{q=0}^{f(p)} \psi_p^a(\xi_{1s})\psi_{pq}^b(\xi_{2t})\hat{u}_{pq} = \sum_{p=0}^{P} \psi_p^a(\xi_{1s}) \left\{ \sum_{q=0}^{f(p)} \psi_{pq}^b(\xi_{2t})\hat{u}_{pq} \right\} \quad \forall s,t. \tag{5.23}$$

In addition, both the summation bound $f(p)$ as the term ψ_{pq}^b of the inner summation now also depends on the index p, which prohibits a formulation similar to (5.15). Instead, the expression above can be written as

$$\boldsymbol{u} = (\boldsymbol{B}_1^a \otimes \boldsymbol{I}_Q)\, \underline{\boldsymbol{B}}^b \hat{\boldsymbol{u}} \tag{5.24}$$

where $\underline{\boldsymbol{B}}^b$ is the block-diagonal concatenation of the matrices $\boldsymbol{B}_p^b$ $(p = 0,\ldots,P)$ which are defined as $\boldsymbol{B}_p^b[i][j] = \psi_{pi}^b(\xi_{2j})$. As opposed to the quadrilateral case, this requires an ordering of the modes $\hat{u}_{n(p,q)}$ where q is running fastest. It can be appreciated that the first sub-step $\underline{\boldsymbol{B}}^b\hat{\boldsymbol{u}}$ cannot be implemented as a matrix-matrix product. Consequently, the implementation of (5.24) consist of $P+1$ matrix-vector multiplications to evaluate $\boldsymbol{v} = \underline{\boldsymbol{B}}^b\hat{\boldsymbol{u}}$, followed by a matrix-matrix multiplication to evaluate $\boldsymbol{u} = (\boldsymbol{B}_1^a \otimes \boldsymbol{I}_Q)\,\boldsymbol{v}$ as $\boldsymbol{U} = \boldsymbol{B}_1^a\boldsymbol{V}^\top$ (In this last step we have chosen $\boldsymbol{U} = \boldsymbol{B}_1^a\boldsymbol{V}^\top$ over $\boldsymbol{U} = \boldsymbol{V}\boldsymbol{B}_1^{a\top}$ in order to ensure an ordering of $u_{r(i,j)}$ with the index i running fastest).

The inner product and mass matrix operators for triangular expansions can be factorised in a similar way. This is discussed in more detail in Appendix A.

5.1.2 The local matrix approach

The second elemental evaluation strategy is the *local matrix* approach. In this approach, the bi-linear form (5.5) is evaluated using the matrix associated with it. The evaluation of the elemental mass matrix operator then reduces to the matrix-vector multiplication

$$\hat{\boldsymbol{y}} = \boldsymbol{M}^e\hat{\boldsymbol{u}}, \tag{5.25}$$

where $\boldsymbol{M}^e[i][j] = \int_{\Omega_e} \phi_i^e\phi_j^e d\Omega_e$ is the elemental mass matrix. For the scope of this work, we do not consider the construction of the mass matrix to be part of the

evaluation. We will assume that $\boldsymbol{M}^e$ has been precomputed (e.g. by means of (5.12) or (5.20)) and is readily available to evaluate (5.25).

5.1.3 The global matrix approach

As opposed to both the previous evaluation strategies, the global matrix approach acts directly upon the *global* bi-linear form (5.2). For the mass matrix operator, this form is then evaluated using the global mass matrix $\boldsymbol{M}$ as

$$\hat{\boldsymbol{y}}_g = \boldsymbol{M}\hat{\boldsymbol{u}}_g, \tag{5.26}$$

where $\boldsymbol{M}[i][j] = \int_\Omega \Phi_i \Phi_j d\Omega$. The finite element method typically leads to global matrices which are very *sparse*. For efficient storage and manipulation of sparse matrices, different formats only storing the non-zero entries of $\boldsymbol{M}$ have been proposed. Again, we assume that the global matrix $\boldsymbol{M}$ has been precomputed (e.g. through the global assembly procedure (2.61)) such that the global matrix evaluation only consist of the (sparse) matrix-vector multiplication.

5.2 Theoretical cost estimates based on operation count

In order to assess the efficiency of the different strategies, we will first analyse the operation count associated to each approach by investigating how many floating point operations each evaluation strategy requires.

5.2.1 Sum-factorisation versus local matrices

When comparing both the elemental strategies for evaluating the quadrilateral mass matrix, it can be observed that the factorised evaluation replaces the matrix-vector multiplication (5.25) by a series of matrix-matrix multiplications, i.e. Eq. (5.20). However, while in the local matrix approach the single matrix $\boldsymbol{M}^e$ is of size $\mathcal{O}(P^2) \times \mathcal{O}(P^2)$, the matrices $\boldsymbol{B}_m^a$ in the sum-factorisation approach are only of size $\mathcal{O}(P) \times \mathcal{O}(P)$. This can also be explained as follows: the elemental mass

matrix $\boldsymbol{M}^e$ truly is a two-dimensional operator while on the other hand, the sum-factorisation technique exploits the tensorial nature of expansion by applying the one-dimensional operators $\boldsymbol{B}^a_m$ along all the lines of constant ξ_n. As a result, the local matrix approach requires $\mathcal{O}(P^4)$ floating point operations to evaluate while the factorised evaluation only involves $\mathcal{O}(P^3)$ operations per matrix-matrix multiplication. This effectively is the strength of the sum-factorisation approach: it replaces an $\mathcal{O}(P^4)$ operation by multiple $\mathcal{O}(P^3)$ operations. This ensures a substantial performance benefit for the limit $P \to \infty$. However for low-order expansions the coefficients of the leading order terms in the operation count do play an important role such that an exact operation count is required to assess whether the sum-factorisation technique truly reduces the number of floating point operations. A complete operation count analysis for four different finite element operators can be found back in Appendix A, and the results are summarised in Table 5.1 and Figures 5.2 and 5.3. From these data, it appears that for the backward transformation and the inner product operator – which for the factorised evaluation only involve two matrix-matrix multiplications – the sum-factorisation technique is always more efficient, except for the linear finite element case ($P = 1$) on triangular elements. However, for more complex operators such as the mass matrix operator or the weak Helmholtz operator defined as

$$a_e(\phi_n, \phi_m) = \int_{\Omega_e} \phi_n(\boldsymbol{x})\phi_m(\boldsymbol{x}) + \lambda \nabla\phi_n(\boldsymbol{x}) \cdot \nabla\phi_m(\boldsymbol{x}) d\boldsymbol{x}, \tag{5.27}$$

the factorised evaluation respectively requires four and eight matrix-matrix multiplications. Consequently, the sum-factorisation technique only becomes more efficient from respectively $P = 5$ and $P = 10$ in the quadrilateral case. For triangular expansions, the break-even point may even be as high as $P = 27$ for the Helmholtz operator.

From the same data, it can also be observed for all operators that the sum-factorisation technique leads to a smaller reduction for triangular than for quadrilateral elements, both in terms of the break-even point as the reduction factor. This reduced effectiveness can be attributed to the triangular tensorial expansion not being constructed as a full tensor product.

		sum-factorisation	local matrix	break-even
Quad	bwd transf	$4P^3 + 18P^2 + 26P + 12$	$2P^4 + 12P^3 + 26P^2 + 24P + 8$	$P = 1$
	inner product	$4P^3 + 19P^2 + 30P + 16$	$2P^4 + 12P^3 + 26P^2 + 24P + 8$	$P = 1$
	mass matrix	$8P^3 + 37P^2 + 56P + 28$	$2P^4 + 8P^3 + 12P^2 + 8P + 2$	$P = 5$
	Helmholtz	$16P^3 + 93P^2 + 184P + 124$	$2P^4 + 8P^3 + 12P^2 + 8P + 2$	$P = 10$
Tri	bwd transf	$3P^3 + 12P^2 + 17P + 8$	$P^4 + 6P^3 + 13P^2 + 12P + 4$	$P = 2$
	inner product	$3P^3 + 13P^2 + 20P + 10$	$P^4 + 6P^3 + 13P^2 + 12P + 4$	$P = 2$
	mass matrix	$6P^3 + 25P^2 + 37P + 18$	$0.5P^4 + 3P^3 + 6.5P^2 + 6P + 2$	$P = 11$
	Helmholtz	$14P^3 + 69P^2 + 113P + 58$	$0.5P^4 + 3P^3 + 6.5P^2 + 6P + 2$	$P = 27$

Table 5.1: Theoretical operation count (floating point multiplications and additions) for the elemental evaluation strategies.

Since Orszag's work (Orszag 1980) in 1980, the sum-factorisation technique has always been considered the key to the efficient implementation of global spectral methods (where a polynomial order $P = 100$ is deemed normal). However, the analysis in this section indicates that this cannot be considered as generally valid for the spectral element method in our chosen operational regime of $1 \leq P \leq 15$. The results show that sum-factorisation does not necessarily lead to a reduction in floating point operations, especially when evaluating complex operators for low-order expansions.

5.2.2 Global matrices versus local matrices

5.2.2.1 Quadrilateral expansions

For a multi-elemental spectral/*hp* expansion, it can be understood from the previous section and Appendix A that the local matrix evaluation will require $2|\mathcal{E}|(P+1)^4$ floating point operations. On the other hand when adopting the global matrix strategy together with a sparse storage format to store $\boldsymbol{M}$, the operation count will scale like nnz, where nnz is the number of non-zero entries in $\boldsymbol{M}$. An entry $\boldsymbol{M}[i][j]$ is typically non-zero if the global basis function Φ_i and Φ_j are coupled, i.e. they have overlapping support. Estimates for the operation count are derived in Appendix A.5 and the results are shown in Table 5.2 and Figure 5.2.

The global matrix evaluation appears to be attractive in particular for low-order elements. In case $P = 1$ every global mode of a structured quadrilateral mesh

		local matrix	global matrix	$\lim_{\|\mathcal{E}\|\to\infty} \frac{\text{GlobMat}}{\text{LocMat}}$
Quad	DOFs	$\|\mathcal{E}\|(P+1)^2$	$\left(\|\mathcal{E}^{1d}\|P+1\right)^2$	$\left(1-\frac{1}{P+1}\right)^2$
	operation count	$2\|\mathcal{E}\|(P+1)^4$	$2\left(\|\mathcal{E}^{1d}\|P(P+2)+1\right)^2$	$\left(1-\frac{1}{(P+1)^2}\right)^2$
Tri	DOFs	$\frac{1}{2}\|\mathcal{E}\|(P+1)(P+2)$	$\frac{1}{2}\|\mathcal{E}\|P^2$	$\frac{P^2}{(P+1)(P+2)}$
	operation count	$2\|\mathcal{E}\|\left(\frac{1}{2}(P+1)(P+2)\right)^2$	$2\|\mathcal{E}\|\frac{1}{4}\left(P^4+6P^3+7P^2\right)$	$\frac{P^4+6P^3+7P^2}{((P+1)(P+2))^2}$

Table 5.2: Operation count estimates of the local matrix and global matrix approach to evaluate a global bi-linear form on a structured quadrilateral mesh ($|\mathcal{E}^{1d}| \times |\mathcal{E}^{1d}| = |\mathcal{E}|$ elements) and an unstructured triangular mesh ($|\mathcal{E}|$ elements).

is coupled to its nine neighbouring global modes (including itself), leading to nine multiply-add pairs per global DOF. However, every global mode corresponds to four local modes which each are coupled to the four linear elemental modes. As a result, the local matrix evaluation then requires $4 \times 4 = 16$ multiply-add pairs per global DOF. This implies that for $P = 1$ using global matrices only requires a fraction $9/16 = 0.5625$ of the floating point operations needed for the local matrix approach (assuming a sufficient mesh size).

When increasing P, relatively more interior modes than edge modes will be added to each element. As there exist a one-to-one mapping between elemental interior modes and global expansion modes, the effect of the multiplicity of the elemental boundary modes will decrease. As a results, when neglecting the cost of assembly of the local matrix approach, the complexity of the global matrix evaluation will asymptotically approach to the complexity of the local matrix evaluation for $P \to \infty$. This can also be observed in Figure 5.2.

5.2.2.2 Triangular expansions

For unstructured triangular meshes, it is more difficult to estimate the non-zero entries of the global matrix $\boldsymbol{M}$ as it depends on the distribution of the triangles within the mesh. Estimates have been made in (Bagheri, Scott, & Zhang 1994) and the results are presented in Table 5.2 and depicted in Figure 5.3. A similar trend as for structured quadrilateral meshes can be observed, but the advantage of the global matrix approach is even greater in the triangular case. This can be explained by

the larger boundary modes to interior modes ratio for triangles, resulting in greater dominance of the multiplicity of global DOFs.

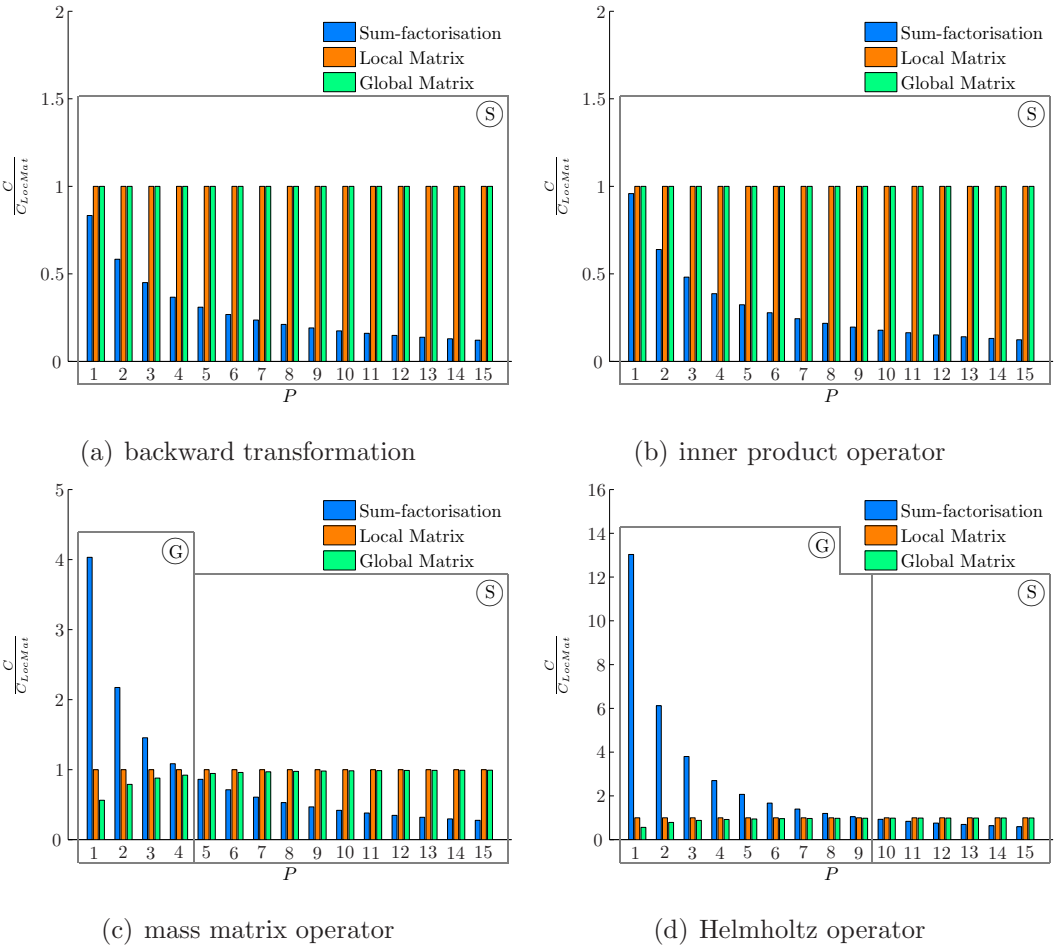
Remark 5 *Although the Helmholtz and mass matrix operators share an identical operation count for both the local and global matrix approach, their results differ from the backward transformation and inner product operators. This is because one dimension of the backward transformation and inner product is in terms of the quadrature points, which are not being assembled. The global matrix evaluation then only benefits from assembly in one direction. However, it appears that this assembly does not lead to a reduction in non-zero entries such that the local matrix approach and global matrix operation have similar operation counts for these operators. On the other hand, the assembly does decrease the rank of the operator.*

5.2.3 Optimal strategy

Based upon the operation count estimates summarised in Figures 5.2 and 5.3, one may conclude that for low-order expansions the global matrix evaluation strategy is superior, while for high-order expansions, the sum-factorisation technique is preferred.

5.3 Computational cost based on run-time

The previous section indicates that the application of different evaluation strategies depending on P leads to a reduction in operation count. Here, we analyse whether this reduction in operation count leads to more efficient algorithms by directly comparing the resulting run-time. This may not be guaranteed as the efficiency of a certain implementation is not only quantified by the number of floating point operations. Various other factors such as the number of memory references, memory access time, caching effects, data structures and chip architecture do play an important role as well. Furthermore the role of possibly optimised linear algebra packages such as BLAS (Dongarra, Du Croz, Hammarling, Hanson, & Duff 1988) should not be forgotten. As it is a cumbersome, if not impossible, task to estimate the cost of



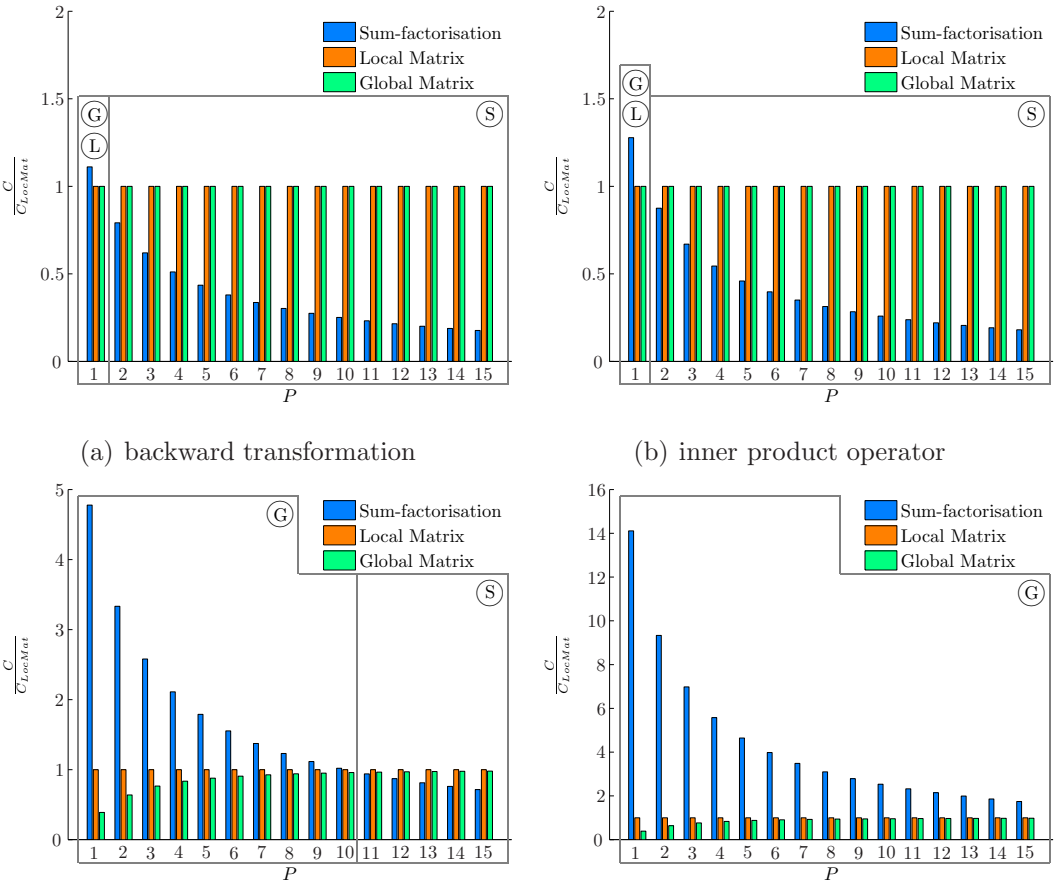
(a) backward transformation

(b) inner product operator

(c) mass matrix operator

(d) Helmholtz operator

Figure 5.2: Operation count results (scaled by the local matrix evaluation operation count) of the different evaluation strategies for a structured quadrilateral mesh. The boxes with encircled tags S (sum-factorisation), L (local matrix) or G (global matrix) indicate the optimal strategy for the corresponding range of P.



(a) backward transformation

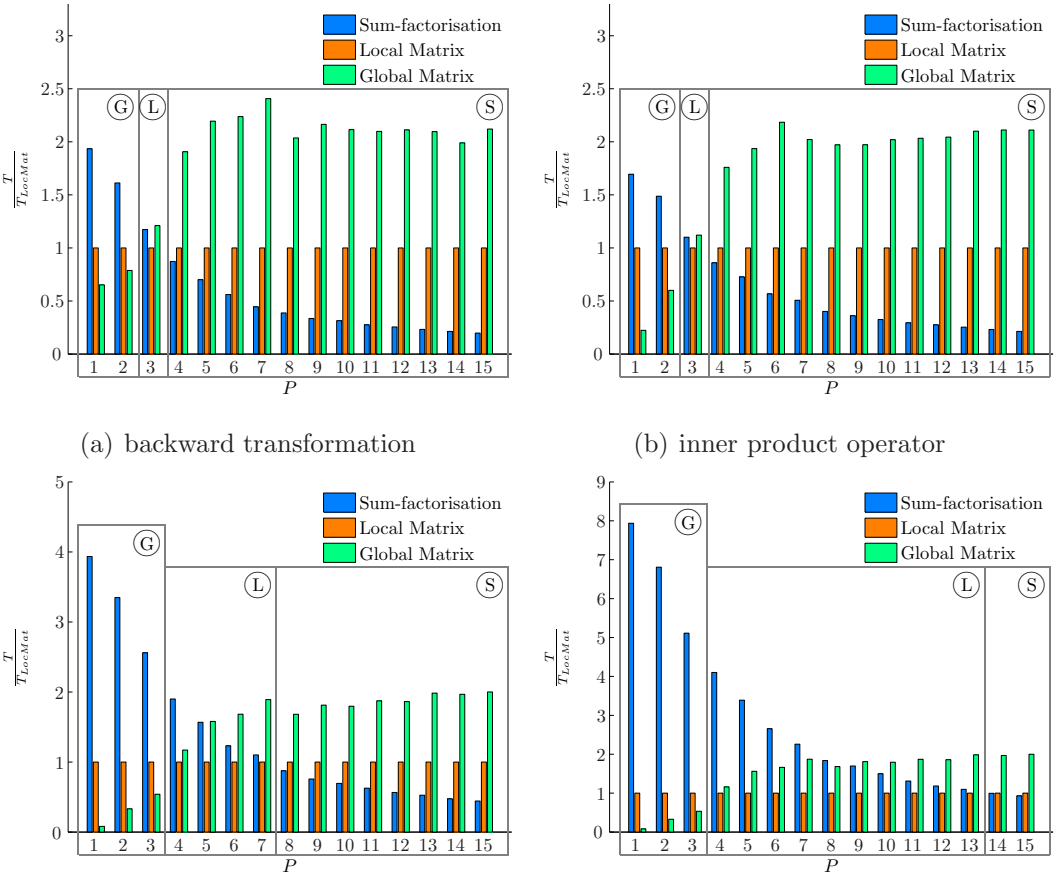
(b) inner product operator

(c) mass matrix operator

(d) Helmholtz operator

Figure 5.3: Operation count results (scaled by the local matrix evaluation operation count) of the different evaluation strategies for an unstructured triangular mesh. The boxes with encircled tags S (sum-factorisation), L (local matrix) or G (global matrix) indicate the optimal strategy for the corresponding range of P.

these separate parameters, the efficiency of the different strategies will be assessed by comparing the total computational cost (quantified by the actual run-time) of the different strategies.



(a) backward transformation

(b) inner product operator

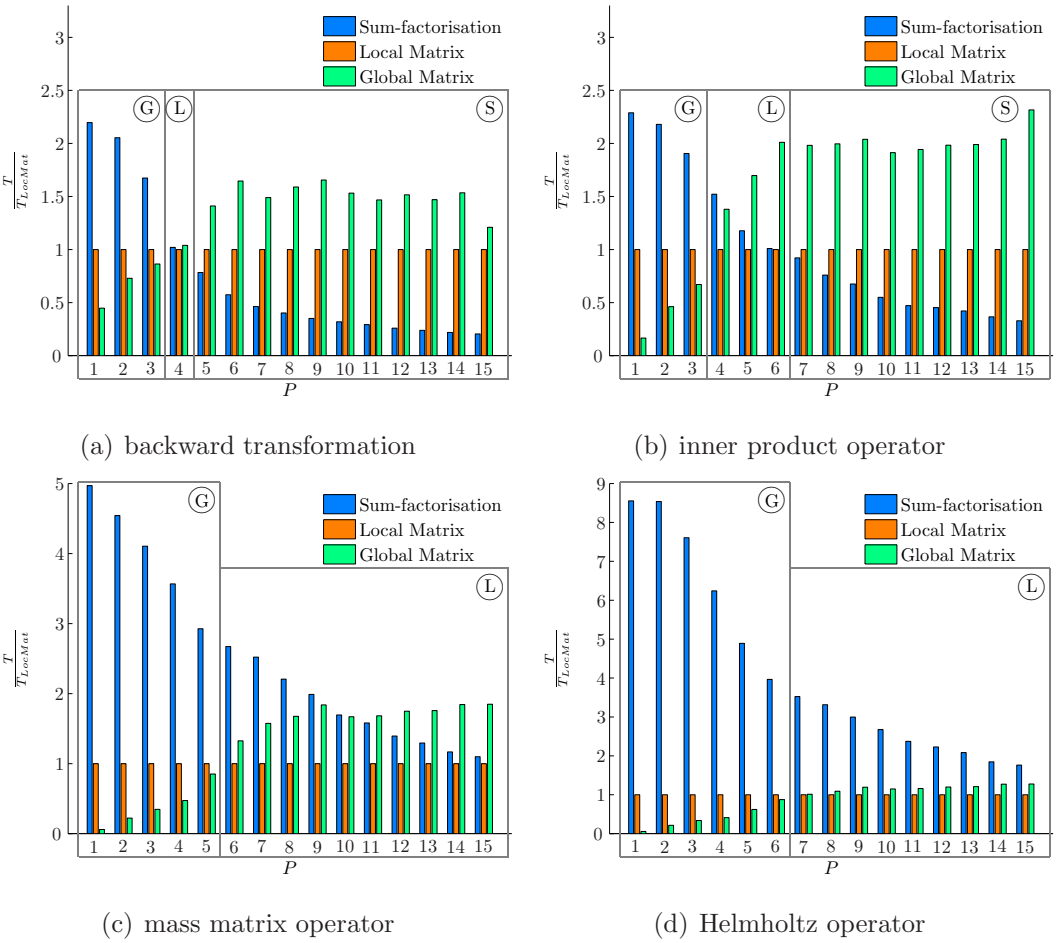
(c) mass matrix operator

(d) Helmholtz operator

Figure 5.4: Computational cost (i.e. run-time scaled by the local matrix evaluation run-time) of the different evaluation strategies for a structured quadrilateral mesh of 1000 elements. The boxes with encircled tags S (sum-factorisation), L (local matrix) or G (global matrix) indicate the optimal strategy for the corresponding range of P.

The results are summarised in Figures 5.4 and 5.5. We will now compare these results with Figures 5.2 and 5.3 respectively leading to the following two important observations:

- The performance benefit of the sum-factorisation technique is reduced, shifting the break-even point between the elemental approaches to the right in favour of



(a) backward transformation

(b) inner product operator

(c) mass matrix operator

(d) Helmholtz operator

Figure 5.5: Computational cost (i.e. run-time scaled by the local matrix evaluation run-time) of the different evaluation strategies for an unstructured triangular mesh of 1032 elements. The boxes with encircled tags S (sum-factorisation), L (local matrix) or G (global matrix) indicate the optimal strategy for the corresponding range of P.

the local matrix approach. This can probably be explained by the fact that the sum-factorisation technique requires temporary storage, which induces some additional cost from a computational point of view.

- Although the global matrix technique is still preferable for low-orders, it rapidly becomes excessively expensive for higher-orders. This is most likely due to the inability to exploit the locality of the data which cancels the reduction in operation count.

Both these factors contribute to the rise of an intermediate regime between low- and high-order expansions, where the local matrix approach now is the optimal strategy. This is clearly visible in both Figure 5.4 and Figure 5.5

All tests presented in this section were run on an Intel MacBook Pro (2.33 GHz dual core processor, 2GB RAM) and the performance tests were based upon the implementation of the different strategies within the Nektar++ framework. The computational kernel for the sum-factorisation technique and local matrix approach was based upon the reference implementation of the BLAS routines `dgemm` and `dgemv` respectively (Dongarra, Du Croz, Hammarling, Hanson, & Duff 1988). The global matrix evaluation was implemented using the `dcsrmm` routine of the NIST sparse BLAS library (Remington & Pozo 1996). Validation tests were run on machines with different specifications, and although the results may differ slightly in terms of the break even points, the general trends and principles observed have been confirmed.

5.4 Lessons learned

In order to efficiently implement the spectral/*hp* element method for a broad range of polynomial orders, one should distinguish three different regimes:

- a low-order regime where the global matrix approach is most efficient,
- an intermediate-order regime where the local matrix approach is most efficient, and
- a high-order regime where the sum-factorisation approach is most efficient.

Note that efficiency is defined here in a computational sense, i.e. minimizing the actual run-time. Remember that we have assumed the use of tensorial expansion bases in the analysis. For non-tensorial expansions such as the nodal triangular spectral/*hp* expansions (Hesthaven & Warburton 2008), only the first two regimes will be applicable as the sum-factorisation technique cannot be applied. Consequently, non-tensorial expansion will not benefit from the observed performance gain due to the sum-factorisation technique in case of high-order expansions.

It can be observed that selecting a non-optimal evaluation strategy can lead to very inefficient code. When for example evaluating the Helmholtz operator for a 4^{th}-order triangular spectral/*hp* expansion, applying the sum-factorisation technique rather than the global matrix approach will increase the run-time by a factor 15. For a linear expansion, this factor has even been observed to be as high as 150.

The break-even points between the different regimes will in general depend on:

- the operator to be evaluated,
- the shape of the element (quadrilateral versus triangle), and
- the computer on which the code is run.

Although the theoretical operation count may have given an indication of these regimes, the results in Section 5.3 show the importance of performance tests to determine the computer-specific break-even points, especially when operating in the intermediate regime between low-order and high-order. For example, based upon operation count, Figure 5.2(c) would suggest that sum-factorisation is the optimal technique to evaluate the mass matrix for a 5^{th}-order expansion. However, Figure 5.4(c) shows that the application of the local matrix approach leads to a reduction in run-time with a factor 1/3. This also supports the idea of a self-tuning library (such as the ATLAS (Whaley, Petitet, & Dongarra 2001) implementation of BLAS): in order to obtain optimal performance, a set of tests should be run during installation in order to determine the machine-specific break-even points.

Chapter 6

Multi-level Static Condensation

As outlined in Section 1.1, an important step in the solution process of the advection-diffusion equation consists of solving the linear system due to the implicit treatment of the elliptic diffusion operator. In this chapter, we will focus on different *direct* solution strategies of such linear systems, with a special emphasis on the static condensation technique and *substructuring*. These techniques may be applied to any general non-symmetric matrix system, but we will here introduce the different concepts through the example of the Helmholtz matrix.

6.1 Static condensation

Assume we have to solve a linear system of the form

$$\boldsymbol{H}\boldsymbol{u} = \boldsymbol{f}, \tag{6.1}$$

where $\boldsymbol{u} = \hat{\boldsymbol{u}}_g$ is the vector of unknown global coefficients (for simplicity, we will omit the circumflex to denote a vector of coefficients in this chapter) and $\boldsymbol{H}$ is the global Helmholtz matrix. Like most system matrices due to a finite element discretisation, $\boldsymbol{H}$ is typically very sparse, but it may have a full bandwidth. It is therefore very inefficient, if not impossible, to store the full matrix so that it can be directly inverted. One possibility is to follow an iterative solution technique based upon the elemental interpretation of the global Helmholtz matrix. Another possibility is to

use direct solution techniques that exploit the structure of the spectral/*hp* element discretisation. In this chapter, we will elaborate on this latter approach.

The static condensation technique is based upon the boundary-interior decomposition of the spectral/*hp* expansion basis. If we order the global boundary degrees of freedom in the vector $\hat{\boldsymbol{u}}_g$ (that is, those constructed from the local boundary modes) first, followed by the global interior degrees (that is, an element-by-element ordering of the elemental interior modes), the vectors $\boldsymbol{u}$ and $\boldsymbol{f}$ can be decomposed as

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_B \\ \boldsymbol{u}_I \end{bmatrix}, \quad \boldsymbol{f} = \begin{bmatrix} \boldsymbol{f}_B \\ \boldsymbol{f}_I \end{bmatrix}, \tag{6.2}$$

where the indices B and I respectively refer to boundary and interior. Then Eq. (6.1) can be written in its constituents parts as

$$\begin{bmatrix} \boldsymbol{H}_{BB} & \boldsymbol{H}_{BI} \\ \boldsymbol{H}_{IB} & \boldsymbol{H}_{II} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_B \\ \boldsymbol{u}_I \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_B \\ \boldsymbol{f}_I \end{bmatrix}, \tag{6.3}$$

The foundation of the static condensation technique now lays in the observation that, due to the chosen ordering of coefficients, the interior-interior matrix $\boldsymbol{H}_{II}$ is block-diagonal, as can be seen in Fig. 6.1 where the matrix $\left[\begin{smallmatrix} \boldsymbol{H}_{BB} & \boldsymbol{H}_{BI} \\ \boldsymbol{H}_{IB} & \boldsymbol{H}_{II} \end{smallmatrix}\right]$ is represented as $\left[\begin{smallmatrix} \boldsymbol{M}_b & \boldsymbol{M}_c \\ \boldsymbol{M}_c^\top & \boldsymbol{M}_i \end{smallmatrix}\right]$. This arises from the fact that the elemental interior modes are non-overlapping and are therefore orthogonal at elemental level. As a result, $\boldsymbol{H}_{II}$ can be inverted elementally, i.e. block-by-block, which is a cheap operation.

In order to apply the static condensation technique, we perform a block elimination by pre-multiplying Eq. (6.3) with

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1} \\ 0 & \boldsymbol{I} \end{bmatrix}, \tag{6.4}$$

to arrive at

$$\begin{bmatrix} \boldsymbol{H}_{BB} - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{H}_{IB} & 0 \\ \boldsymbol{H}_{IB} & \boldsymbol{H}_{II} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_B \\ \boldsymbol{u}_I \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_B - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1} f_I \\ \boldsymbol{f}_I \end{bmatrix}. \tag{6.5}$$

The system can now be solved by first solving the first block-row for the boundary degrees of freedom, followed by solving the second block-row for the interior degrees of freedom. This essentially comes down to a solution procedure consisting of the following four steps:

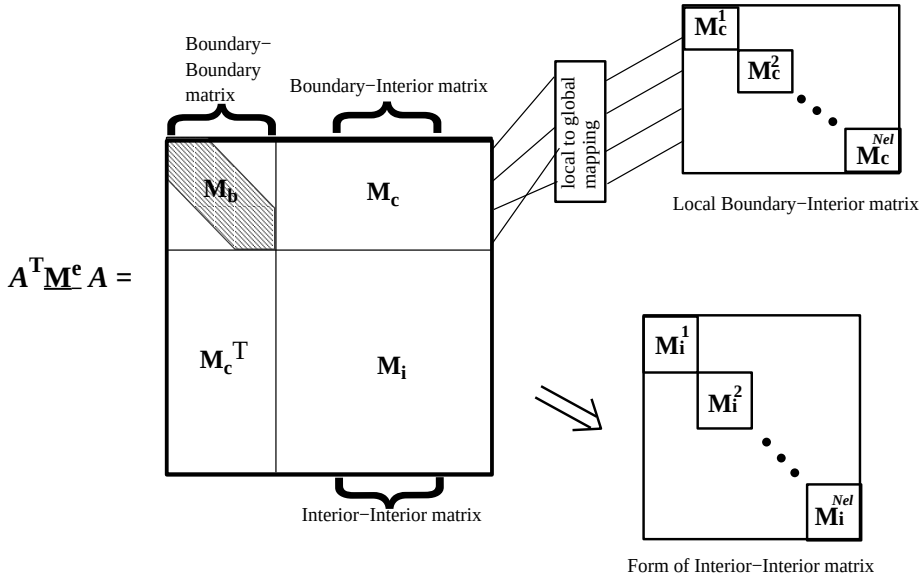


Figure 6.1: Structure of the global matrix system when the degrees of freedom are ordered appropriately in order to apply the static condensation technique. Courtesy of (Karniadakis & Sherwin 2005).

1. Calculate the modified right hand-side vector $\boldsymbol{g}_B$

$$\boldsymbol{g}_B = \boldsymbol{f}_B - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{f}_I. \tag{6.6}$$

2. Solve

$$\boldsymbol{S}\boldsymbol{u}_B = \boldsymbol{g}_B, \tag{6.7}$$

for the boundary degrees of freedom $\boldsymbol{u}_B$, where the matrix $\boldsymbol{S} = \boldsymbol{H}_{BB} - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{H}_{IB}$ is known as the *Schur complement*.

3. Calculate the modified right hand-side vector $\boldsymbol{g}_I$

$$\boldsymbol{g}_I = \boldsymbol{f}_I - \boldsymbol{H}_{IB}\boldsymbol{f}_B. \tag{6.8}$$

4. Solve

$$\boldsymbol{H}_{II}\boldsymbol{u}_I = \boldsymbol{g}_I, \tag{6.9}$$

for the interior degrees of freedom $\boldsymbol{u}_I$ by evaluating $\boldsymbol{u}_I = \boldsymbol{H}_{II}^{-1}\boldsymbol{g}_I$.

The only four matrices needed in this static condensation solution procedure can be collected together as

$$\begin{bmatrix} \boldsymbol{S} & \boldsymbol{R} \\ \boldsymbol{H}_{IB} & \boldsymbol{H}_{II}^{-1} \end{bmatrix}, \tag{6.10}$$

where $\boldsymbol{R} = \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}$. Thus in order to solve Eq. 6.1 we only need to construct and store this matrix system above (this maybe does not hold for the Schur complement for which we somehow need the inverse but that will be the subject of the following sections). Furthermore, it is possible not to work with the global interpretation of the matrices $\boldsymbol{R}$ and $\boldsymbol{H}_{IB}$, but only to store and use the respective elemental equivalents (see also Fig. 6.1)

- $\underline{\boldsymbol{R}^e} = \left[\underline{\boldsymbol{H}_{BI}^e \left[\boldsymbol{H}_{II}^e\right]^{-1}}\right]$, which is the block-diagonal concatenation of the product of the elemental matrices $\boldsymbol{H}_{BI}^e$ and $\left[\boldsymbol{H}_{II}^e\right]^{-1}$, and
- $\underline{\boldsymbol{H}_{IB}^e}$, which is the block-diagonal concatenation of the elemental matrices $\boldsymbol{H}_{IB}^e$.

Note that the notation of an underlined matrix refers to the block diagonal extension of elemental matrices. It can also be appreciated that the global interior-interior matrix $\boldsymbol{H}_{II}^{-1}$ is equal to its elemental equivalent $\underline{\left[\boldsymbol{H}_{II}^e\right]}^{-1}$ as the global and elemental interpretations do coincide in this case. The matrix operators $\boldsymbol{R}$ and $\boldsymbol{H}_{IB}$ can then be evaluated elementally, respectively as

$$\boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1} f_I = \boldsymbol{\mathcal{A}}_B^{\top} \underline{\boldsymbol{H}_{BI}^e} \boldsymbol{H}_{II}^{-1} f_I, \tag{6.11}$$

$$\boldsymbol{H}_{IB}\boldsymbol{f}_B = \underline{\boldsymbol{H}_{IB}^e} \boldsymbol{\mathcal{A}}_B f_B, \tag{6.12}$$

where $\boldsymbol{\mathcal{A}}_B$ and $\boldsymbol{\mathcal{A}}_B^{\top}$ respectively are the boundary scatter operator (from global boundary degrees of freedom to local boundary degrees of freedom) and boundary assembly operator (from local boundary degrees of freedom to global boundary degrees of freedom).

The fact that three out of four matrices can be stored and evaluated elementally makes the static condensation technique very effective for the spectral/*hp* element technique. As a result, the majority of the storage requirement and computational effort is needed for the global Schur complement system. This can be further optimised by either using bandwidth minimising techniques such as the Cuthill-McKee algorithm or by adopting a multi-level static condensation approach. This will be elaborated in the remainder of this chapter.

6.2 Multi-level static condensation

Once we have condensed out the elemental interior modes of the spectral/*hp* expansion, we can apply the same idea of static condensation to the remaining boundary degrees of freedom in order to solve the Schur complement system. In the field of structural mechanics, this idea of multi-level static condensation is also known as *substructuring* (Smith, Bjorstad, & Gropp 1996). Recently, a few articles have been published (Elssel & Voss 2008; Gao, Xiaoye, Chao, & Zhaojun 2008; Rachowicz & Zdunek 2009) that study the concept of substructuring in the context of sparse eigenproblems. To facilitate the introduction of the multi-level static condensation technique, consider Fig. 6.2(a) where we have given a graphical interpretation of the global boundary modes associated to the Schur complement matrix for a structured quadrilateral mesh of $8 \times 8 = 64$ elements. In this figure, one can make a distinction between the boundary vertex modes and the boundary edge modes. We have omitted the degrees of freedom on the boundary of the domain assuming Dirichlet boundary conditions (see also Section 6.5) for a detailed discussion of Dirichlet boundary conditions. Note that for expansion orders of $P > 2$, there will be more than one mode on every edge. But because the connectivity pattern of these multiple edge modes is identical, we will always collect them in a single node of the graph that represents the boundary degrees of freedom and their connectivity.

The idea behind multi-level static condensation is to order the boundary degrees of freedom such that submatrices appear that are block-diagonal after which the static condensation technique can be applied on this new level. To accomplish this, we will have to collect several boundary modes in different patches. The modes that are interior to such a patch are then orthogonal with respect to the *interior modes* of the other patches and the coupling only exists through the degrees of freedom that form the interfaces between the patches. A patch-by-patch numbering of the degrees of freedom will then result in the desired block-diagonal structure. Although such a reordering and division in patches can be intuitively constructed for simple meshes, we aim for an automated procedure that can be applied to unstructured and arbitrary meshes. That is why we will use the nested bisection algorithm from

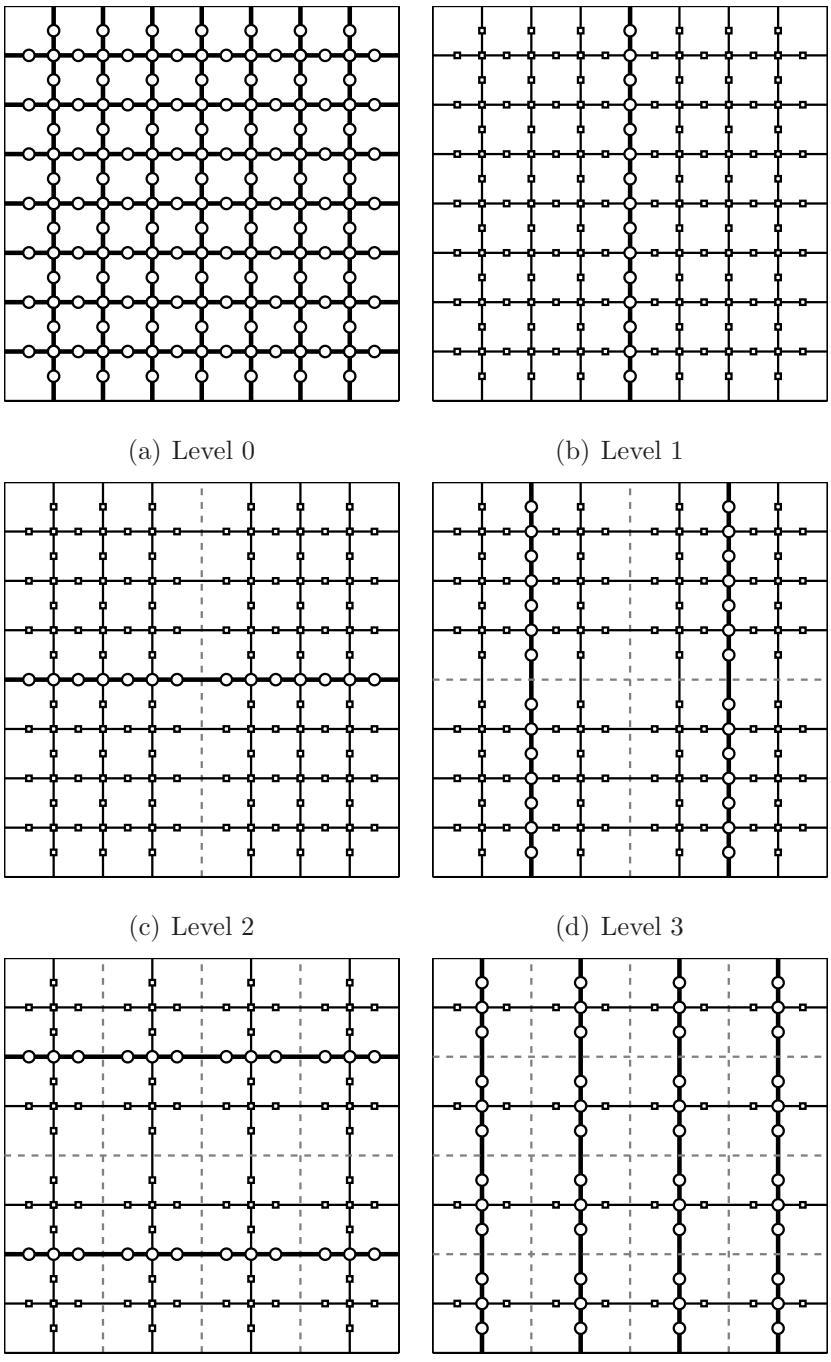
the *Metis* (Karypis & Kumar 1998) graph partitioning package. We will later come back to discuss the advantages of this algorithm, see Section 6.4. This partitioning is based on the graph representation of the boundary modes. All boundary modes are nodes in the corresponding graph and their connectivity (two nodes are connected if their corresponding modes have overlapping support) define the edges of the graph. As such, the sparsity pattern of the resulting system matrix (i.e. the Helmholtz matrix in this case) can also be seen as a graphical interpretation of the graph. As a result, the nested bisection algorithm to partition the graph can be considered as a reordering of the system matrix (and degrees of freedom). It is also important to know that the boundary-boundary matrix $\boldsymbol{H}_{BB}$ and the corresponding Schur complement matrix $\boldsymbol{S}$ share the same graph and, as a result, the same sparsity pattern (that is, if you neglect any orthogonality that may exist in the elemental expansion basis).

Starting from the idea of a nested bisection algorithm, the static condensation technique can be applied in two different ways: following a top-down approach or following a bottom-up approach. We will investigate both approaches in the sections below.

6.2.1 Top-down multi-level static condensation

A single step of the bisection algorithm divides the degrees of freedom in two different patches with an interface in between as can be observed in Fig. 6.2(b). It is customary to consider the (boundary) modes interior to the patch as *interior* degrees of freedom and the (boundary) modes on the interface, or separator, as the *boundary* degrees of freedom. The first step in the top-down approach is to number the interface degrees of freedom first, followed by a patch-by-patch numbering of the interior degrees of freedom. Similar as in Eq. (6.3), the Schur complement out of Eq. (6.7) can then be decomposed as

$$\boldsymbol{S}_0 = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{B}_1 \\ \boldsymbol{C}_1 & \boldsymbol{D}_1 \end{bmatrix}, \tag{6.13}$$



(a) Level 0

(b) Level 1

(c) Level 2

(d) Level 3

(e) Level 4

(f) Level 5

Figure 6.2: The different patches and the corresponding division between boundary and interior modes for the different levels of the top-down multi-level static condensation technique. Given is the example for an 8×8 mesh and an expansion order of $P = 2$.

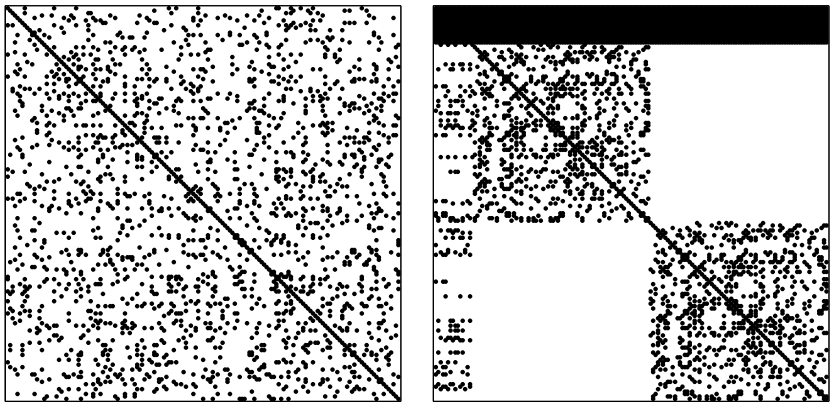
where the subscripts 0 and 1 respectively refer to the level. After applying the static condensation technique on this system, we end up with the following matrices (see also Eq. (6.10)) needed to solve the Schur complement system associated to $\boldsymbol{S}_0$, i.e.

$$\begin{bmatrix} \boldsymbol{S}_1^{-1} & \boldsymbol{R}_1 \\ \boldsymbol{C}_1 & \boldsymbol{D}_1 \end{bmatrix}, \tag{6.14}$$

where the new Schur complement $\boldsymbol{S}_1$ is equal $\boldsymbol{S}_1 = \boldsymbol{A}_1 - \boldsymbol{B}_1\boldsymbol{D}_1^{-1}\boldsymbol{C}_1$ to and $\boldsymbol{R}_1$ can be constructed as $\boldsymbol{R}_1 = \boldsymbol{B}_1\boldsymbol{D}_1^{-1}$. The sparsity pattern of this matrix system (6.14) for the example of Fig. 6.2(b) is given in Fig. 6.3(b) where we can clearly observe a block diagonal structure in the submatrix $\boldsymbol{D}_1$ (consisting of two blocks). Compared to Eq. (6.10), we have deliberately replaced the Schur complement $\boldsymbol{S}_1$ by its inverse $\boldsymbol{S}_1^{-1}$ and $\boldsymbol{D}_1^{-1}$ by $\boldsymbol{D}_1$. This is because in the multi-level top-down approach, we will explicitly calculate the inverse of the Schur complement (due to its limited size) and continue to the next level by applying the static condensation technique to the diagonal blocks in the matrix $\boldsymbol{D}_1$ (in order to solve systems of the type $\boldsymbol{D}_1\boldsymbol{u} = \boldsymbol{f}$). Therefore, both patches of the first level will each have to be split again in the second level. This strategy is displayed in Fig. 6.2 and naturally fits the concept of the nested bisection algorithm. It also allows for a recursive application of the static condensation technique to the interior-interior matrices $\boldsymbol{D}_i$. Fig. 6.3 depicts the sparsity patterns of the corresponding system matrices (6.14) of this top-down approach up to the maximum level of 5 recursions for the example in question. In the last level, the matrices $\boldsymbol{D}_5$ can be directly inverted.

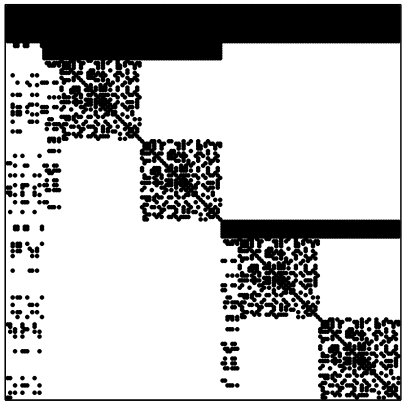
6.2.2 Bottom-up multi-level static condensation

As opposed to the previous approach, this approach starts from the *bottom* of the nested bisection algorithm. It takes all the patches of the last level of the nested bisection as the patches of the first level of the multi-level static condensation technique. This is depicted in Fig. 6.4(b). Again, a distinction can be made between *interior* degrees of freedom and *boundary* degrees of freedom. Note that in this first step, we have a large number of boundary degrees of freedom compared to the first step of the top-down approach. Also note that there are much more patches

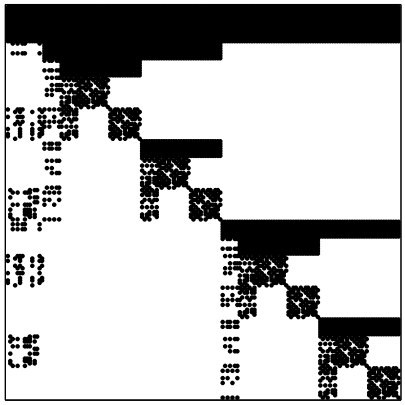


(a) Level 0: matrix $\boldsymbol{S}_0$

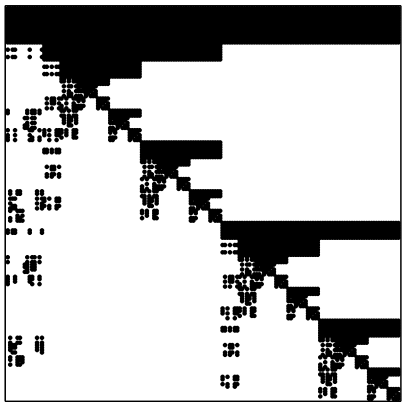
(b) Level 1



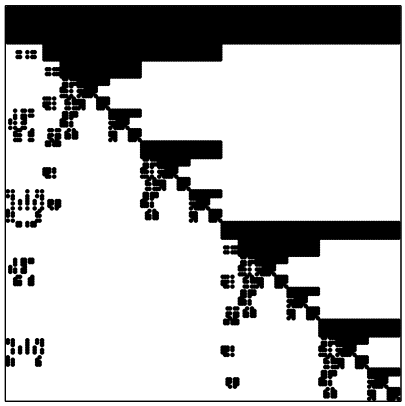
(c) Level 2



(d) Level 3



(e) Level 4



(f) Level 5

Figure 6.3: The sparsity pattern of the matrix systems given by Eq. (6.14) for the top-down multi-level static condensation technique applied to the example of Fig. 6.2.

as compared to the first level of the top-down approach, but that every patch only has one interior mode (remember that this mode may represent more than one degree of freedom in case $P > 2$). Introducing an appropriate ordering of the degrees of freedom where all the boundary degrees of freedom are followed by a patch-by-patch numbering of the interior degrees of freedom allows us to decompose the Schur complement matrix $\boldsymbol{S}_0$ as given by Eq. (6.13)

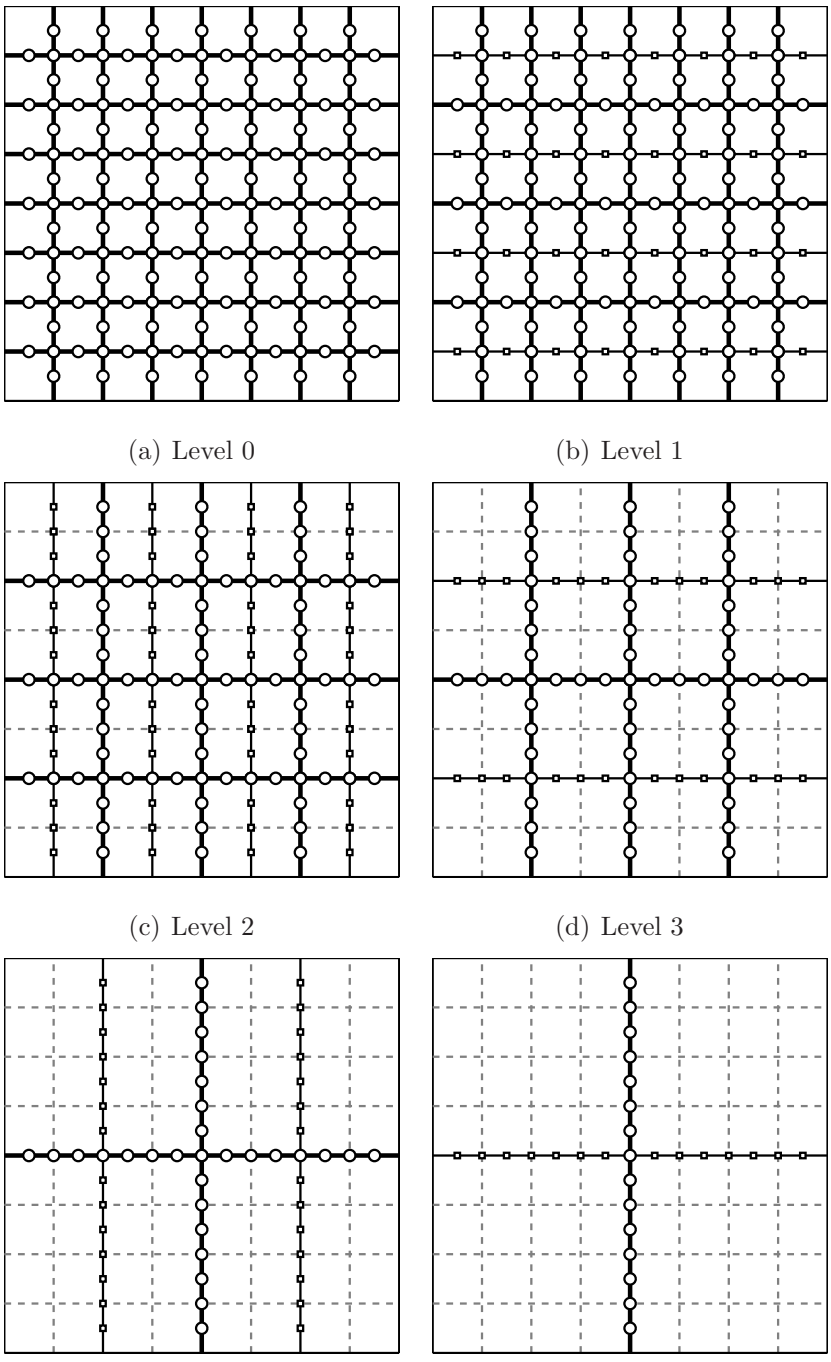
After the application of the static condensation technique, we will need the following matrix system in order to *invert* $\boldsymbol{S}_0$, i.e.

$$\begin{bmatrix} \boldsymbol{S}_1 & \boldsymbol{R}_1 \\ \boldsymbol{C}_1 & \boldsymbol{D}_1^{-1} \end{bmatrix}, \tag{6.15}$$

where $\boldsymbol{S}_1$ and $\boldsymbol{R}_1$ are defined as in the previous section. The sparsity pattern of this matrix is shown in Fig. 6.5(b). Note that we now do save the inverse matrix $\boldsymbol{D}_1^1$ as it can be trivially inverted. The new Schur complement $\boldsymbol{S}_1$ still is big in size and consequently expensive to invert. That is why in the bottom-up approach, we recursively apply the static condensation technique on the Schur complement. On every level, we make use of the nested bisection algorithm to identify the patches and make a distinction between the boundary and the interior degrees of freedom. The complete bottom-up multi-level approach for the example of the regular 8×8 mesh is depicted in Fig. 6.4. The corresponding sparsity patterns of the resulting system matrices are depicted in Fig. 6.5.

6.3 Theoretical cost estimates

In the previous sections, it has been indicated that for the application of the multi-level static condensation technique, you need to evaluate either the matrix depicted in Fig. 6.3(f) (that is, for the top-down approach) or the matrix given in Fig. 6.5(f) (that is, for the bottom-up approach). As the number of floating point operations required to evaluate a sparse matrix scales like the number of non-zero entries in the matrix, it can be appreciated that we can make a theoretical cost estimate of both approaches based upon the fill-in associated to the respective sparsity patterns.

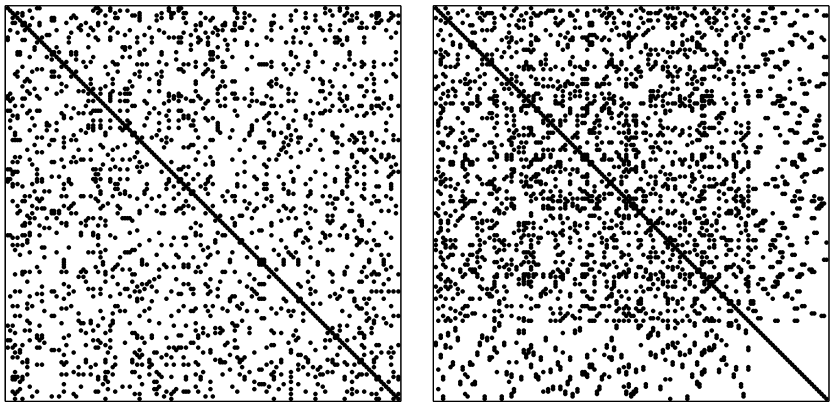


(a) Level 0 (b) Level 1

(c) Level 2 (d) Level 3

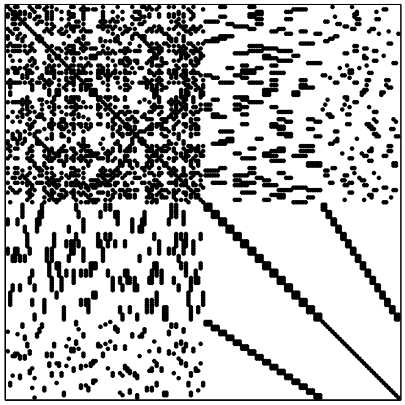
(e) Level 4 (f) Level 5

Figure 6.4: The different patches and the corresponding division between boundary and interior modes for the different levels of the bottom-up multi-level static condensation technique. Given is the example for an 8×8 mesh and an expansion order of $P = 2$.

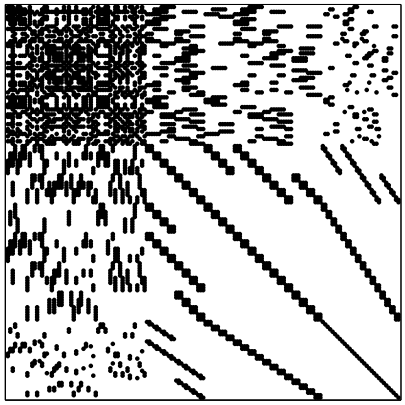


(a) Level 0: matrix $\boldsymbol{S}_0$

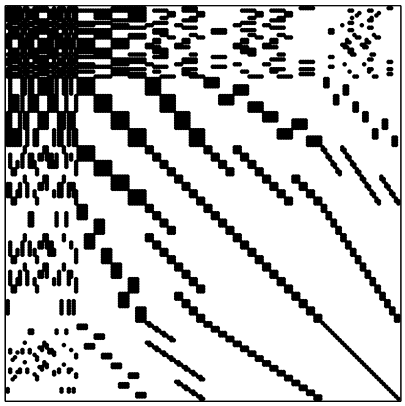
(b) Level 1



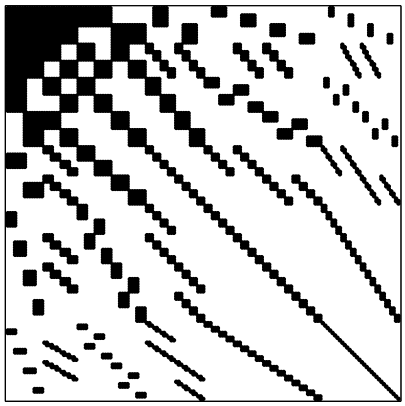
(c) Level 2



(d) Level 3



(e) Level 4



(f) Level 5

Figure 6.5: The sparsity pattern of the matrix systems given by Eq. (6.15) for the bottom-up multi-level static condensation technique applied to the example of Fig. 6.4.

However, it is important to emphasize that this does not necessarily requires the use of sparse matrix storage formats to apply multi-level static condensation. For the boundary-boundary and interior-interior blocks on the diagonal, this is obvious. For the off-diagonal sub-matrices $\boldsymbol{R}_i$ (bottom-up) and $\boldsymbol{C}_i$ (top-down and bottom-up), we can adopt an elemental interpretation of the degrees of freedom similar as in Eq. (6.11) and Eq. (6.12) to circumvent the use of sparse matrix storage. Therefore, we need to consider the patches at a certain level as if they were *elements*. This allows for the use of storage efficient block-diagonal matrices, without increasing the cost as the number of non-zero entries do not change.

We have calculated the number of non-zero entries (denoted by C) in the matrices out of Fig. 6.3(f) and Fig. 6.5(f) and displayed the ratio

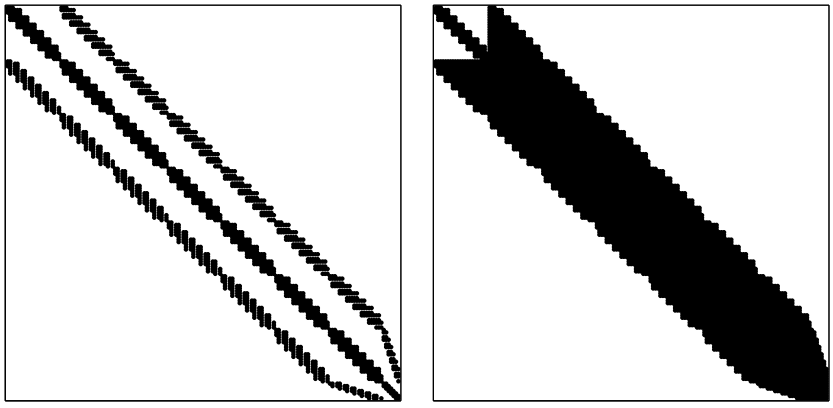
$$\frac{C_{\text{topdown}}}{C_{\text{bottomup}}} \tag{6.16}$$

in Fig. 6.7 in order to theoretically assess the cost associated to both approaches. For comparison, we have also included the solution strategy in which we apply a bandwidth minimisation algorithm to the matrix $\boldsymbol{S}_0$ rather than the multi-level static condensation technique. Fig. 6.6(a) shows the sparsity pattern of $\boldsymbol{S}_0$ in case the degrees of freedom are ordered for minimal bandwidth (which corresponds to a lexicographical ordering for the 8×8 mesh under consideration). As the bandwidth (or the skyline more generally) is preserved when factorising a matrix (but the band may be filled), it can be appreciated that solving the banded system scales like the number of matrix elements within the band. This can also be observed in Fig. 6.6(b) where the sparsity pattern of the matrix $\boldsymbol{L}+\boldsymbol{U}$ is shown, $\boldsymbol{L}$ and $\boldsymbol{U}$ being the result of the LU factorisation of $\boldsymbol{S}_0$.

In Fig. 6.7(a), it can be seen for the 8×8 mesh, the top-down approach and the bandwidth minimisation approach are more expensive than the bottom-up multi-level static condensation technique. The difference is smaller for low expansion orders, but the overhead cost goes to a factor 1.4 and 2 respectively for higher-orders. Note that for orders higher than $P > 2$, the multiplicity of the edge modes comes into play favouring the bottom-up approach. We have done a similar analysis for bigger meshes. A mesh of $2^m \times 2^m$ theoretically allows for $2m - 1$ levels of

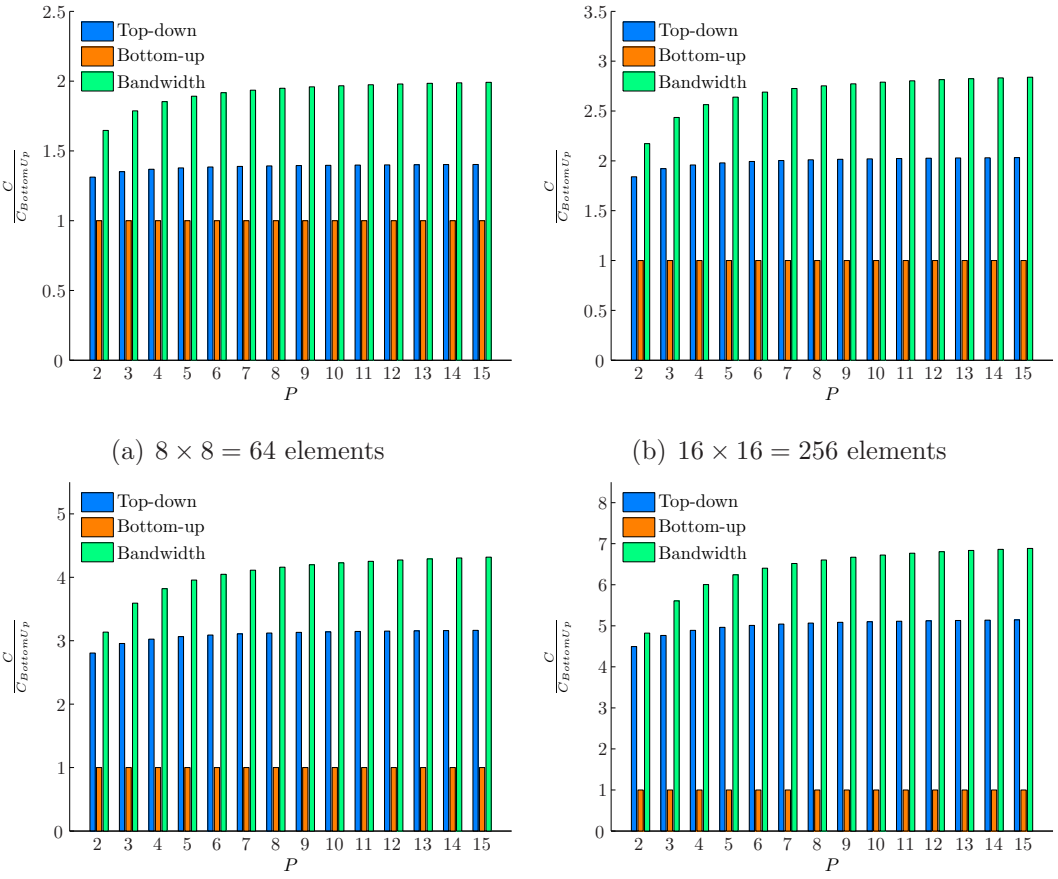
recursions. In Figs. 6.7(b), 6.7(c) and 6.7(d), it can be observed that the cost advantage of the bottom-up technique increases with the number of elements. For a mesh of $64 \times 64 = 4096$ elements, the application of the bottom-up multi-level static condensation technique leads to a theoretical saving up to a factor 5 and 6.5 compared to respectively the top-down technique and the banded matrix approach. Note that this cost comparison for the number of required floating point operations in principal also holds for the storage requirements.

From this theoretical analysis, we conclude that the bottom-up approach is the superior multi-level static condensation technique when compared to the top-down approach. This mainly is due to high cost associated to the complete fill-in of the off-diagonal blocks $\boldsymbol{R}_i$ as can clearly be observed in Fig. 6.3(f). This can be explained by the fact that the largest part of the resulting matrix in the top-down approach basically is a reordering of the original matrix $\boldsymbol{S}_0$. A big part of the coupling between modes is transferred to $\boldsymbol{R}_i$ blocks, hence its density. The bottom-up multi-level approach on the other hand implicitly transfers the coupling information from level to level by every time calculating the Schur complement of the Schur-complement, thereby completely transforming the original matrix. This clearly turns out to be best strategy. Therefore, we will choose the bottom-up approach as the preferred multi-level static condensation in the remainder of this chapter.



(a) matrix $\boldsymbol{S}_0$ (b) LU decomposition of $\boldsymbol{S}_0$

Figure 6.6: The sparsity patterns of the Schur complement $\boldsymbol{S}_0$ and its factorisation after minimisation of the bandwidth for the example mesh of Fig. 6.2(a).



(a) $8 \times 8 = 64$ elements

(b) $16 \times 16 = 256$ elements

(c) $32 \times 32 = 1024$ elements

(d) $64 \times 64 = 4096$ elements

Figure 6.7: Operation count results (scaled by the bottom-up approach) of the different direct solution strategies for structured quadrilateral meshes of $2^m \times 2^m$ elements.

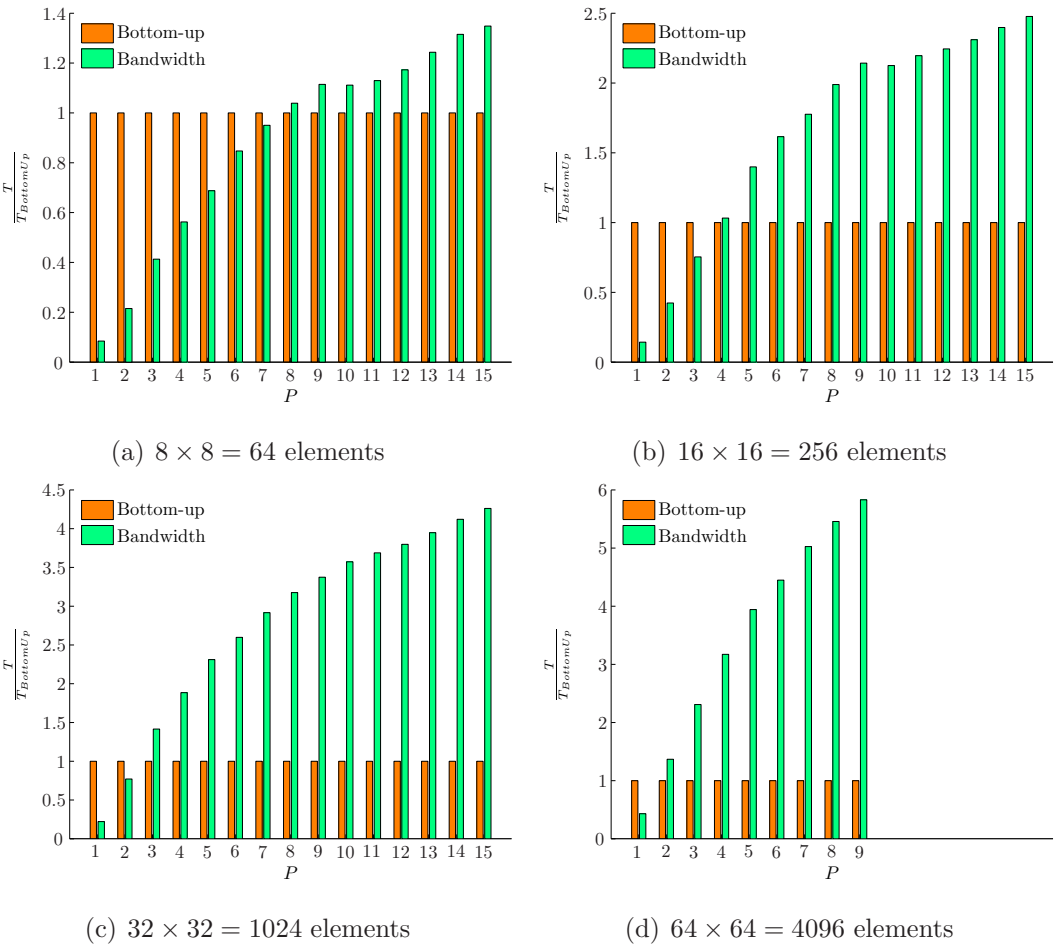
6.4 Computational cost

In this section, we will investigate whether the observed benefit of the *bottom-up* multi-level static condensation technique over the more traditional approach of bandwidth minimisation also persists when implemented within a computer environment. The implementation of the nested bisection algorithm needed to partition the graph is based upon the `onmetis` routine of the Metis software package (Karypis & Kumar 1998). This routine essentially is designed to compute fill-reducing orderings of sparse matrices, which in light of the discussion above, happens to corresponds to our needs. However, we will use a slightly modified version developed by Gao et al.

(Gao, Xiaoye, Chao, & Zhaojun 2008), that in addition to reordering also returns the separator tree. This is needed to construct the different block-matrices at every level of multi-level static condensation technique. The evaluation of the different block matrices is based upon the `dgemv` routine of the BLAS package. We observed that for the 8×8 example mesh, the Metis multi-level bisection algorithm resulted in exactly the same partitioning as the intuitive partitioning of the previous sections.

As in chapter 5, we will only consider the run-time needed to evaluate the statically condensed matrix system, thereby neglecting any computational effort needed to construct the matrices or performing the partitioning. The same holds for the banded matrix solution technique for which we rely on LAPACK (Anderson, Bai, Bischof, Blackford, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney, & Sorensen 1999) routines `dptrf` and `dptrs`.

The results that compare the computational cost (quantified by the measured run-time needed to solve the system associated to $\boldsymbol{S}_0$) of both approaches are shown in Fig. 6.8 for the same quadrilateral meshes as studied in Fig. 6.7. It can be observed that for small meshes, the banded matrix technique is superior for low-orders. This makes sense as these problems only involve a limited number of degrees of freedom and the large amount of function calls associated to the multi-level recursive algorithm clearly affect the run-time (despite the lower operation count). For the more relevant examples of the bigger meshes, we see that multi-level static condensation technique is superior in almost all cases. Only for linear finite elements, the banded matrix technique is faster. For higher-orders, the multi-level static condensation technique may be up to 4.5 (for the mesh of $32 \times 32 = 1024$ elements) to more than 6 (for the mesh of $64 \times 64 = 4096$ elements) times faster. This also indicates why the technique of substructuring is more attractive for high than for low-order elements: the greater multiplicity of the edge modes makes it more efficient to condense out these degrees of freedom. A similar behaviour can be observed when applying the multi-level static condensation technique on unstructured triangular meshes, as can be seen from Fig. 6.9.



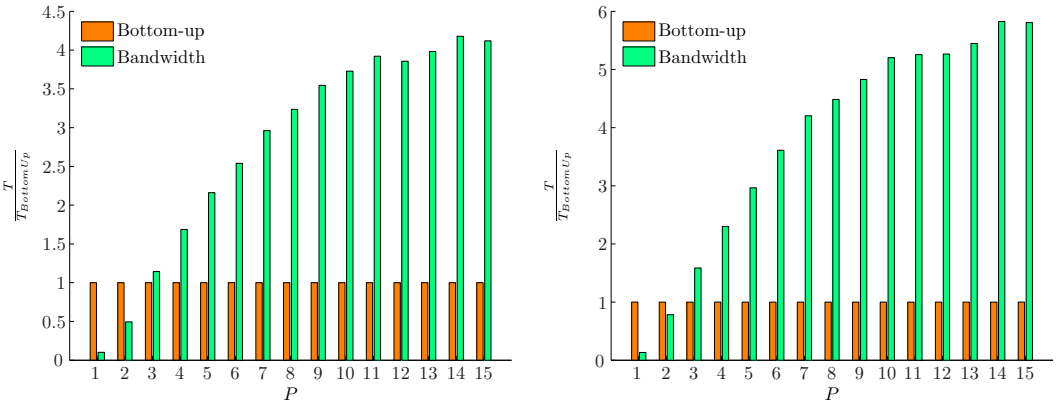
(a) $8 \times 8 = 64$ elements

(b) $16 \times 16 = 256$ elements

(c) $32 \times 32 = 1024$ elements

(d) $64 \times 64 = 4096$ elements

Figure 6.8: Computational cost (i.e. run-time scaled by the bottom-up approach run-time) of the different direct solution strategies for structured quadrilateral meshes of $2^m \times 2^m$ elements.



(a) 1032 elements

(b) 2133 elements

Figure 6.9: Computational cost (i.e. run-time scaled by the bottom-up approach run-time) of the different direct solution strategies for unstructured triangular meshes

6.5 Dirichlet boundary conditions

In case of numerically solving a problem subjected to Dirichlet boundary conditions, it is customary adopt a lifting strategy as for example explained in Section 4.3.1. If in the solution vector $\boldsymbol{u}$, we first number the degrees of freedom associated to the Dirichlet boundary before all other (*homogeneous*) degrees of freedom, the system given by Eq. (6.1) can be decomposed as

$$\begin{bmatrix} \boldsymbol{H}_{DD} & \boldsymbol{H}_{DH} \\ \boldsymbol{H}_{HD} & \boldsymbol{H}_{HH} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_D \\ \boldsymbol{u}_H \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_D \\ \boldsymbol{f}_H \end{bmatrix}, \tag{6.17}$$

where the index D denotes the Dirichlet degrees of freedom and the index H refer to the homogeneous degrees of freedom. This system can be solved for the unknown degrees of freedom $\boldsymbol{u}_H$ by solving the linear system

$$\boldsymbol{H}_{HH}\boldsymbol{u}_H = \boldsymbol{f}_H - \boldsymbol{H}_{HD}\boldsymbol{u}_D. \tag{6.18}$$

The (multi-level) static condensation technique can then be applied to the matrix $\boldsymbol{H}_{HH}$. This is what we have implicitly assumed in Section 6.3

It should be noted that the modification of the right-hand-side can be an expensive operation. It is explained in Section 7.1 that, because the matrix $\boldsymbol{H}_{HD}$ is not

explicitly available in our implementation, we calculate the Dirichlet forcing

$$\boldsymbol{f}_H^{\mathrm{DIR}} = \boldsymbol{H}_{HD}\boldsymbol{u}_D, \tag{6.19}$$

by means of the forward operation of the entire Helmholtz operator $\boldsymbol{H}$, i.e.

$$\begin{bmatrix} \boldsymbol{f}_D^{\mathrm{DIR}} \\ \boldsymbol{f}_H^{\mathrm{DIR}} \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} \boldsymbol{u}_D \\ \boldsymbol{u}_H \end{bmatrix}, \tag{6.20}$$

after which we subtract $\boldsymbol{f}_H^{\mathrm{DIR}}$ from $\boldsymbol{f}_H$. Note that the Helmholtz operation above can be evaluated globally, elementally or using the sum-factorisation approach (see Chapters 5 and 7). However, it is possible to circumvent this rather expensive operation by taking along the Dirichlet degrees of freedom in the (multi-level) static condensation process. To appreciate this point, consider the following decomposed version of system (6.1), i.e.

$$\left[\begin{array}{cc|c} \boldsymbol{H}_{DD} & \boldsymbol{H}_{DB} & \boldsymbol{H}_{DI} \\ \boldsymbol{H}_{BD} & \boldsymbol{H}_{BB} & \boldsymbol{H}_{BI} \\ \hline \boldsymbol{H}_{ID} & \boldsymbol{H}_{IB} & \boldsymbol{H}_{II} \end{array}\right] \left[\begin{array}{c} \boldsymbol{u}_D \\ \boldsymbol{u}_B \\ \hline \boldsymbol{u}_I \end{array}\right] = \left[\begin{array}{c} \boldsymbol{f}_D \\ \boldsymbol{f}_B \\ \hline \boldsymbol{f}_I \end{array}\right], \tag{6.21}$$

where we have separated the Dirichlet boundary modes (index D), the homogeneous boundary modes (index B) and the homogeneous interior modes (index I). We can now apply the static condensation technique on all the boundary modes (that is, Dirichlet and homogeneous boundary modes) to arrive at

$$\left[\begin{array}{cc|c} \boldsymbol{H}_{DD} - \boldsymbol{H}_{DI}\boldsymbol{H}_{II}^{-1}\boldsymbol{H}_{ID} & \boldsymbol{H}_{DB} - \boldsymbol{H}_{DI}\boldsymbol{H}_{II}^{-1}\boldsymbol{H}_{IB} & 0 \\ \boldsymbol{H}_{BD} - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{H}_{ID} & \boldsymbol{H}_{BB} - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{H}_{IB} & 0 \\ \hline \boldsymbol{H}_{ID} & \boldsymbol{H}_{IB} & \boldsymbol{H}_{II} \end{array}\right] \left[\begin{array}{c} \boldsymbol{u}_D \\ \boldsymbol{u}_B \\ \hline \boldsymbol{u}_I \end{array}\right] = \tag{6.22}$$

$$\left[\begin{array}{c} \boldsymbol{f}_D - \boldsymbol{H}_{DI}\boldsymbol{H}_{II}^{-1}\boldsymbol{f}_I \\ \boldsymbol{f}_B - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{f}_I \\ \hline \boldsymbol{f}_I \end{array}\right], \tag{6.23}$$

If only applying a single level of static condensation, the homogeneous boundary modes $\boldsymbol{u}_B$ can be computed by solving the equation

$$\left(\boldsymbol{H}_{BB} - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{H}_{IB}\right)\boldsymbol{u}_B = \boldsymbol{f}_B - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{f}_I - \left(\boldsymbol{H}_{BD} - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{H}_{ID}\right)\boldsymbol{u}_D, \tag{6.24}$$

which, in terms of the Schur complement matrix $\boldsymbol{S}$, can be written more concise as

$$\boldsymbol{S}_{BB}\boldsymbol{u}_B = \boldsymbol{f}_B - \boldsymbol{H}_{BI}\boldsymbol{H}_{II}^{-1}\boldsymbol{f}_I - \boldsymbol{S}_{BD}\boldsymbol{u}_D. \tag{6.25}$$

Note that compared to Eq. (6.6), the right-hand-side contains the additional Dirichlet forcing term $\boldsymbol{f}_B^{\text{DIR}} = \boldsymbol{S}_{BD}\boldsymbol{u}_D$. Again, because the matrix $\boldsymbol{S}_{BD}$ may not be available explicitly, it is possible to evaluate this Dirichlet forcing as

$$\begin{bmatrix} \boldsymbol{f}_D^{\text{DIR}} \\ \boldsymbol{f}_B^{\text{DIR}} \end{bmatrix} = \boldsymbol{S} \begin{bmatrix} \boldsymbol{u}_D \\ \boldsymbol{u}_B \end{bmatrix}. \tag{6.26}$$

This Schur complement matrix $\boldsymbol{S}$ can be significantly smaller than the original system matrix $\boldsymbol{H}$ needed to calculate the Dirichlet forcing as in Eq. (6.20), leading to a corresponding performance benefit. The other steps in the static condensation solution procedure are unaffected by this approach. Because we have adopted an elemental evaluation strategy for the off-diagonal blocks, see Eqs. (6.11) and (6.12), there will be no difference in operation count. It is only the local-to-global mapping that should be slightly modified to include the Dirichlet degrees of freedom.

This type of treatment of the Dirichlet boundary conditions can be recursively applied in case of the multi-level static condensation technique. The Dirichlet boundary conditions will then only have to be transferred to the right-hand-side as a forcing function at the innermost level. And due to the limited size of the Schur complement at the last level, it can be appreciated that this is a relatively cheap operation compared to the original calculation of the Dirichlet forcing at the top-level, as given by Eq. (6.20). Therefore, we can conclude that recursively incorporating the Dirichlet boundary conditions in the application of the multi-level static condensation technique is the preferred approach.

6.6 Concluding remarks

Finally we would like to finish this chapter with some concluding remarks.

- Although the performance focus in this work was mainly on operation count and execution time, we would like to remark that the theoretical analysis of

Section 6.3 also holds for memory storage. Therefore, bottom-up multi-level static condensation technique can also be considered as the optimal approach in that sense.

- The multi-level static condensation technique (that is, the prevailing bottom-up approach) on its innermost level is reduced to a relatively small Schur complement to be inverted. Considering Fig. 6.4(f), it can be appreciated that this Schur complement corresponds to the degrees of freedom of the interface in the first step of the nested bisection algorithm. This means that the multi-level static condensation technique essentially reduces the original two-dimensional problem to a one-dimensional problem. However, it should be emphasised that this one-dimensional system to be inverted is fully coupled. This explains why the multi-level static condensation technique can be expected to be more efficient for two-dimensional problems than for three dimensional problems. In 3D, the original three-dimensional problem will be reduced to a fully coupled two-dimensional system which still can be too expensive to solve (at least directly). This means that the lower the dimensionality of the problem, the bigger the savings that can be expected from the multi-level static condensation technique.

- In relation to the previous remark, it can be appreciated that the multi-level static condensation technique may be more efficient for meshes of a slender domain (such as e.g. for pipe-flows). The small number of elements in one dimension of the mesh allow for bisecting interfaces that are small in size. This implies that even for the innermost levels of recursion, the subsystems can be limited in size (even for three-dimensional problems).

Chapter 7

The optimal spectral/*hp* element discretisation

Now we have determined the most efficient implementation for both low- and high-order expansions, we may ask the following question: Given the most efficient implementation, what is the *optimal spectral/*hp *discretisation* for a given error tolerance? We define the optimal discretisation as the (h, P)-pair – the specific combination of mesh size h and polynomial order P – which requires the minimal run-time to approximate the exact solution up to a predefined accuracy. Answers to this question have been presented before in (Rønquist 1988; Fischer & Gottlieb 1996; Hesthaven 1997; Hesthaven 1998; Wasberg & Gottlieb 2000). In these previous works, the computational cost of the algorithms have been estimated by analytical models. However, based upon the results of the previous section, we believe that an analysis from a purely analytic point of view may not be able to correctly model all factors that make up the actual run-time. Therefore, we adopt a fully computational approach where we base our analysis on the measured run-time of a set of performance test. In addition, note that we do not aim to provide a universal statement to answer this question as it will highly depend of the problem under consideration. Instead we have identified a few examples and demonstrate how the results of the previous section may influence the discussion. Therefore, we have chosen to solve the scalar Helmholtz equation for four different test cases.

7.1 Test problem: the scalar Helmholtz equation

The two-dimensional scalar Helmholtz problem on a domain Ω is given by the equation

$$u(x,y) - \lambda \nabla^2 u(x,y) = f(x,y), \quad \lambda \geq 0. \tag{7.1}$$

The problem is supplied with Dirichlet boundary conditions on the entire boundary of the domain, i.e. $u(x,y)|_{\partial\Omega} = g_{\mathcal{D}}(x,y)$ and we for simplicity we assume $\lambda = 1$. We have chosen not to consider a more complex test case as we believe the simplicity of this problems enables us to unambiguously investigate the influence of the mesh-size h and polynomial order P. If for example selecting a time-dependent problem such as the advection-diffusion equation, the time-step Δt will depend on the mesh-size h through the CFL-condition. It can be appreciated that this additional dependency will quickly complicate the analysis.

To solve Eq. (7.1) we follow a standard Galerkin procedure together with a lifting strategy to impose the Dirichlet Boundary conditions similar as in Section 4.3.1 to arrive at the discrete system in terms of the unknown homogeneous coefficients $\hat{\boldsymbol{u}}^H$,

$$\boldsymbol{H}^{HH}\hat{\boldsymbol{u}}^H = \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W}\boldsymbol{f} - \boldsymbol{H}^{HD}\hat{\boldsymbol{u}}^D, \tag{7.2}$$

where $\boldsymbol{H}$ is the Helmholtz matrix with entries

$$\boldsymbol{H}^{HD}[i][j] = \int_\Omega \Phi_i^H \Phi_j^D + \nabla\Phi_i^H \cdot \nabla\Phi_j^D d\boldsymbol{x} \quad i \in \mathcal{N}^H, j \in \mathcal{N}^D. \tag{7.3}$$

In order to obtain the answer in physical space rather than in coefficient space, we can perform a backward transformation on the result above, yielding

$$\begin{aligned} \boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}} &= \begin{bmatrix} \boldsymbol{B}^D & \boldsymbol{B}^H \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{bmatrix}, \\ &= \begin{bmatrix} \boldsymbol{B}^D & \boldsymbol{B}^H \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}^D \\ \left(\boldsymbol{H}^{HH}\right)^{-1}\left\{\left(\boldsymbol{B}^H\right)^\top \boldsymbol{W}\boldsymbol{f} - \boldsymbol{H}^{HD}\hat{\boldsymbol{u}}^D\right\} \end{bmatrix}. \end{aligned} \tag{7.4}$$

From an implementational point of view, Eq. (7.4) can typically be solved in four major steps (thereby neglecting the steps involving linear combinations of vectors):

1. Calculate the inner product of the forcing function, i.e. $\left(\boldsymbol{B}^H\right)^\top \boldsymbol{W}\boldsymbol{f}$.

2. Calculate the Dirichlet forcing, i.e. $\boldsymbol{H}^{HD}\hat{\boldsymbol{u}}^{D}$.

3. Solve the linear system $\boldsymbol{H}^{HH}\hat{\boldsymbol{u}}^{H} = \hat{\boldsymbol{f}}$.

4. Transform the coefficients back to physical space, i.e. $\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}$

Note that because the operators $\left(\boldsymbol{B}^{H}\right)^{\top}$ and $\boldsymbol{H}^{HD}$ are not explicitly available within the Nektar++ framework, we have adopted an implementation strategy where these operators are evaluated using the equivalent operators in terms of all the global degrees of freedom. As a result, steps 1 and 2 above are evaluated as $\boldsymbol{B}^{\top}\boldsymbol{W}\boldsymbol{f}$ and $\boldsymbol{H}\hat{\boldsymbol{u}}$ respectively where after we only consider the homogeneous part of the solution vector. Consequently, steps 1,2 and 4 in the solution procedure above respectively correspond to the inner product operator, the weak Helmholtz operator and the backward transformation as defined in Chapter 5. This implies that these three sub-steps of the solution procedure may benefit from the different implementation strategies introduced in Chapter 5. It is therefore through these routines that the influence of the different implementation strategies on the definition of optimal *hp*-discretisations may become apparent. Finally note that for the implementation of the third step is based upon the (single-level) static condensation technique with bandwidth minimisation.

7.2 Test problem 1: Quadrilateral spectral/*hp* discretisations for a smooth solution

The first example we consider is a smooth solution on the unit square $\Omega = [0,1] \times [0,1]$. The forcing function f and the Dirichlet boundary conditions $g_{\mathcal{D}}$ are chosen such that the exact solution satisfies

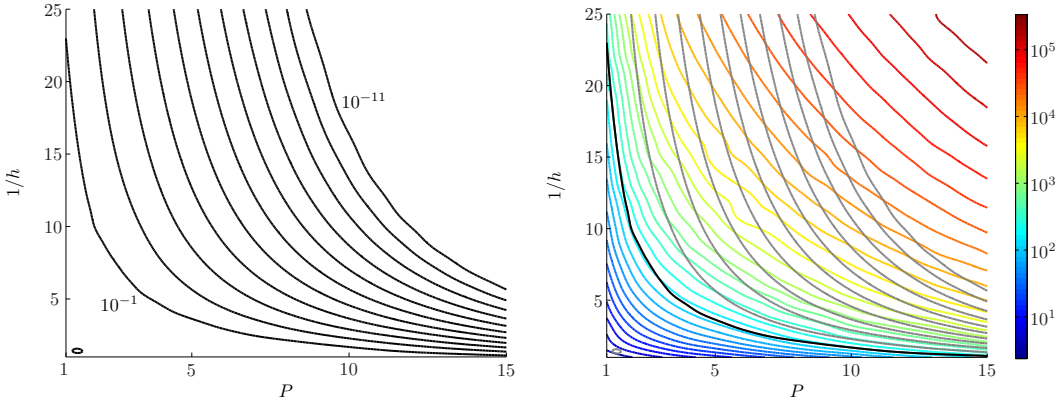
$$u_{ex}(x,y) = \sin(10\pi x)\cos(10\pi y). \tag{7.5}$$

Our aim is to find the quadrilateral *hp*-discretisation which minimises the run-time for a certain error-tolerance. Therefore we let the mesh-size between and the polynomial order respectively vary between $1 \leq 1/h = |\mathcal{E}^{1d}| \leq 25$, $1 \leq P \leq 15$ and solve

the corresponding Helmholtz equation for all 375 possible (h, P) combinations. Note that the spatial uniformity of the solution justifies the uniform adaptation of the expansion. The L_2 approximation error defined as

$$||\epsilon||_{L_2} = \left[\int_\Omega (u_{ex} - u)^2 \, d\boldsymbol{x}\right]^{\frac{1}{2}}, \tag{7.6}$$

is depicted in Figure 7.1(a) for all discretisations. Predictably, low-order expansions on a coarse mesh exhibit a large error while high-order expansions on a fine mesh lead to the most accurate results. The computational cost – quantified by the run-time – of every (h, P) pair is plotted in a similar style in Figure 7.1(b) where the error plots are overlaid. We here show the run-time result based upon the optimal



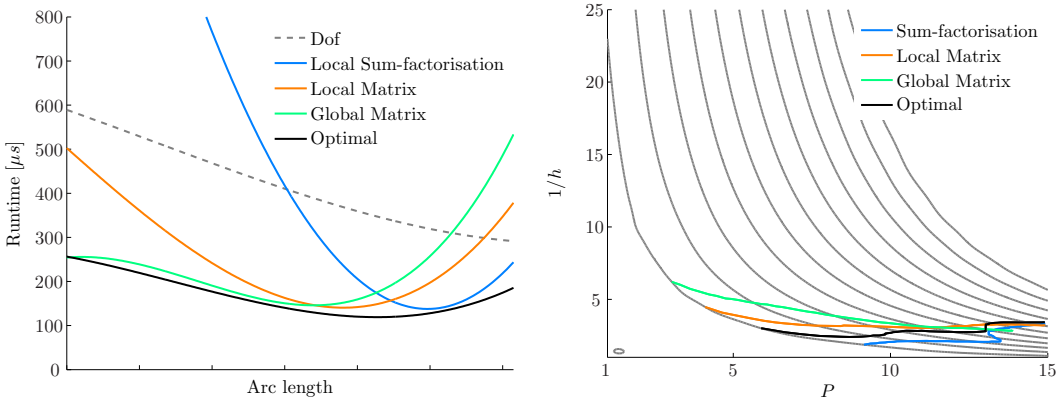
(a) Error contours $||\epsilon||_{L_2} = 10^i \, (1 \leq i \leq 11)$ (b) Run-time contours (μs)

Figure 7.1: Error and run-time of the different quadrilateral spectral/hp discretisations used for approximating the Helmholtz problem with smooth exact solution (7.5).

implementation, i.e. an implementation where depending on the polynomial order P we have selected the most efficient evaluation strategy for each individual operator following the results of Section 5.3. Although both the error and cost contours follow a similar trend, they are not parallel. Note that although all data are essentially discrete, we have interpolated the data in the presentation of these results to obtain a continuous representation.

If we now fix the error-tolerance to 10%, we can ask how these plots may help us to find the optimal discretisation (h, P) satisfying this tolerance? Therefore,

we want to know which point on the 10% error contour line depicted in Figure 7.1(a) induces the minimal run-time. Therefore we can use the arc-length of this contour line as the horizontal axis and plot the corresponding computational cost from Figure 7.1(b) as a function of this arc-length, i.e. we extract the data along the solid black line in 7.1(b). This is shown in Figure 7.2(a). Besides the cost due



(a) Run-time along 10% error contour

(b) Path of minimal run-time

Figure 7.2: Minimal run-time at fixed error-level of the different quadrilateral spectral/*hp* discretisations used for approximating the Helmholtz problem with smooth exact solution (7.5).

to the optimal implementation, this figure includes different lines all corresponding to a different interpretation of the computational cost. The dashed line defines the computational cost as the total number of global degrees of freedom while the other lines are due to a single implementation strategy using either a global matrices, local matrices or the sum-factorisation approach. Note that all four lines lead to a different minimum and that the one due to the optimal implementation truly minimises the run-time of all the different implementation strategies. Table 7.1 (Test problem 1) summarises the *hp*-discretisations to which these minima relate back. It appears that *hp*-expansion employing 3×3 elements and a sixth-order expansion can be regarded as the optimal discretisation for our computational test. Note that this expansion is notably different from the one that minimises the number of degrees of freedom for the given error tolerance. For the example under consideration, this would suggest a spectral-type approach employing only a single element but with

		Test problem 1		Test problem 2		Test problem 3	
		$\lvert\mathcal{E}\rvert$	P	$\lvert\mathcal{E}\rvert$	P	$\lvert\mathcal{E}\rvert$	P
minimal run-time	Optimal implementation	3×3	$P = 6$	6×6	$P = 6$	38	$P = 6$
	Sum-factorisation	2×2	$P = 9$	4×4	$P = 9$	10	$P = 11$
	Local Matrix	5×5	$P = 4$	10×10	$P = 4$	22	$P = 8$
	Global Matrix	6×6	$P = 3$	12×12	$P = 3$	84	$P = 4$
minimal degrees of freedom		1×1	$P > 15$	1×1	$P > 15$	22	$P = 8$

Table 7.1: Quadrilateral spectral/*hp* discretisations to approximate the Helmholtz problem with smooth exact solution (7.5) within 10% accuracy in minimal run-time.

a sufficiently high-order $P > 15$. As expected from Section 5, an implementation based upon global matrices has a tendency towards *h*-type low-order finite elements while the use of sum-factorisation based routines would advocate a more spectral approach.

Figure 7.2(a) also illustrates the importance of supporting different implementation strategies within a single code. As the curve due to the optimal implementation strategy could be considered as the envelope of the three single-implementation curves, selecting another *hp*-discretisation along the horizontal axis does not drastically increases the computational cost. However, adopting only a single implementation strategy it can be seen that the optimum is much sharper and deviating from this optimal discretisation is severely penalised. Perhaps, another performance indicator that may be of interest to consider is the computational performance in flops as it would highlight to which extent a certain implementation strategy exploits all the available computer resources.

In Figure 7.3, we again have plotted the run-time due to the optimal implementation strategy along the 10% error contour. In addition, we have indicated which fraction of the run-time is due to the solution of the linear system, i.e. the third step in the solution process of the Helmholtz equations as explained in Section 7.1. It can be observed that along this error contour, solving the linear system is more efficient for the expansions that combine fewer elements with higher P. This may be a direct consequence of the smaller number of global degrees of freedom as observed in Figure 7.2(a). As a result, it are the three remaining steps in the evaluation process

which shift the minimum towards a lower-order expansion employing a more refined mesh. This behaviour will probably be reflected when using an iterative rather than direct solver.

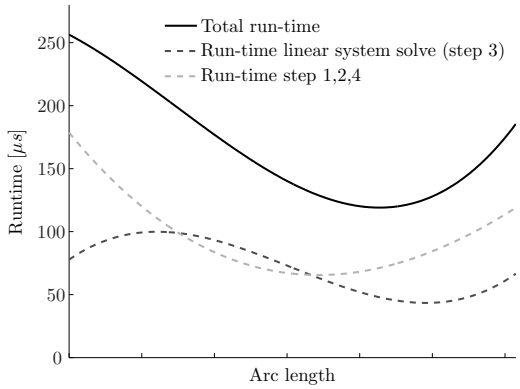


Figure 7.3: Decomposition of the run-time due to the optimal (hybrid) implementation strategy along the 10% error contour when approximating the Helmholtz problem with smooth exact solution (7.5).

We can follow a similar analysis for a broad range of error tolerances. This leads to the *path of minimal run-time* as depicted in Figure 7.2(b). This figure does confirm the well-known feature of exponential p-convergence of the spectral/hp element method for smooth solutions. When aiming for a high-accurate approximation, one should increase the polynomial order rather than refining the mesh. It appears that for this example, a 3×3 mesh is the optimal h-discretisation and the polynomial order P should be varied according to the desired accuracy. This typical behaviour has been acknowledged before and it is widely appreciated that the spectral/hp element method is particularly efficient for obtaining high accuracy in smooth problems (Deville, Fischer, & Mund 2002; Karniadakis & Sherwin 2005). However, by focusing on a engineering accuracy of 10%, we have shown that the high-order methods can also be justified for relatively low accurate approximations.

Finally, we would like to couple this path of minimal run-time to an earlier observation made by Gottlieb and Orszag. In (Gottlieb & Orszag 1977), they showed that for a one-dimensional problem with exact solution

$$u(x) = \sin(M\pi x) \qquad \text{on } [-1, 1], \tag{7.7}$$

a single-element spectral expansion should retain at least $N > M\pi$ modes in order to achieve exponential p-convergence. Translated to the two-dimensional problem under consideration, this would suggest that for exponential p-convergence, the multi-elemental spectral/hp element discretisation should satisfy

$$N > 10\,\pi\,\frac{h}{2}, \tag{7.8}$$

where $N = P+1$. This condition is graphically represented in Figure 7.4 as the area above the *Gottlieb-Orszag threshold*. Heuristically, this may lead to the following discretisation strategy: use h-type refinement until crossing this *Gottlieb-Orszag threshold* and subsequently increase the polynomial order P according to the desired accuracy. This leads to the convergence path also shown in Figure 7.4. Although this approach may provide a simple rule of thumb, the results show that the resulting convergence path is different from the path of minimal run-time defined earlier and hence will be computationally less efficient. The problem is that in this approach, it has been assumed that one should follow an h-type refinement strategy (with $P = 1$) in the unresolved regime. Although this indeed leads to the least expensive point on the Gottlieb-Orszag threshold on Figure 7.4, this point certainly does not minimise the run-time on the corresponding error contour. Instead, our analysis shows that for minimal run-time one should combine both ideas of h-refinement and p-enrichment to select an initial discretisation along the Gottlieb-Orszag threshold. Hereafter, the polynomial order P can be increased for higher accuracy.

7.3 Test problem 2: Quadrilateral spectral/*hp* discretisations for a smooth solution with high wave-number

As a second example, we consider the Helmholtz equation with a similar solution but with a higher wave-number

$$u(x, y) = \sin(20\pi x)\cos(20\pi y). \tag{7.9}$$

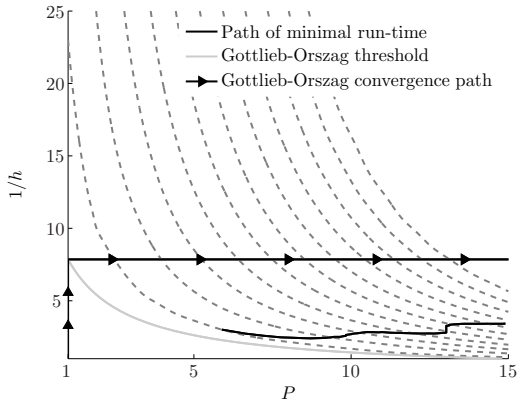
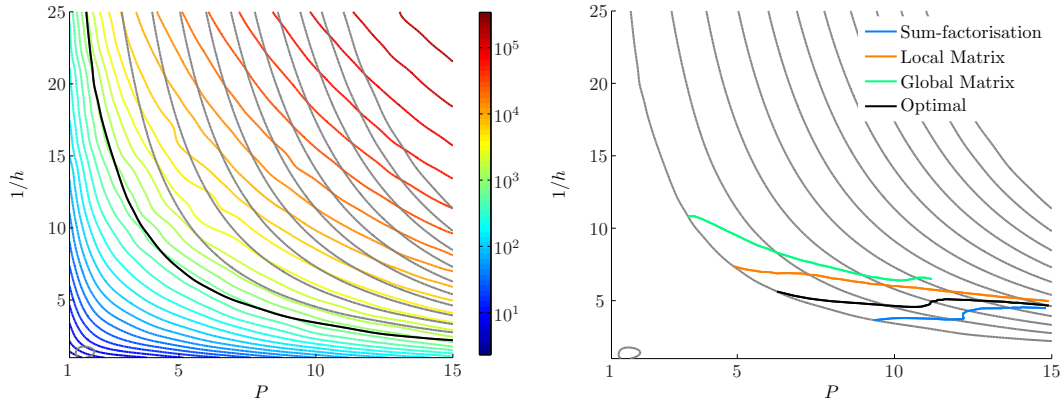


Figure 7.4: Possible path of convergence when following the argument presented by Gottlieb and Orszag when approximating the Helmholtz problem with smooth exact solution (7.5).

Obviously, the computational cost for each specific spectral/*hp* expansion is the same as in the previous example but it now will require more degrees of freedom to obtain a specific error tolerance as can be seen from Figure 7.5(a). Following a



(a) Run-time (μs) and error contours $||\epsilon||_{L_2} = 10^i\,(1 \leq i \leq 11)$

(b) Path of minimal run-time

Figure 7.5: Quadrilateral spectral/*hp* discretisations to approximate the Helmholtz problem with smooth exact solution (7.9).

similar approach as in the previous example, it can be derived that for an engineering accuracy of 10%, the optimal *hp*-discretisation minimising the run-time again comprises a sixth-order expansion but now on a 6×6 mesh, see also Table 7.1 and Fig. 7.5(b) . As in the previous case, the most efficient way of enhancing the accuracy is by increasing the polynomial order and keeping the mesh fixed. The analogy

with the previous example seems to suggest that for the type of exact solution under consideration, i.e.

$$u(x, y) = \sin(M\pi x)\cos(M\pi y), \tag{7.10}$$

the optimal quadrilateral h-discretisation consists of $\frac{3M}{10} \times \frac{3M}{10}$ elements, independent of the desired accuracy. Hence, it is the polynomial order that should be varied accordingly to satisfy a predefined error tolerance.

7.4 Test problem 3: Triangular spectral/*hp* discretisations for a smooth solution

Next we consider exactly the same problem as in Section 7.2 for the first test problem. However, rather then using structured quadrilateral meshes we now use unstructured triangular meshes. A typical unstructured triangular mesh with $h = 1/20$ is shown in Figure 7.6. Figure 7.7(a) depicts the error and cost contours lines across the entire

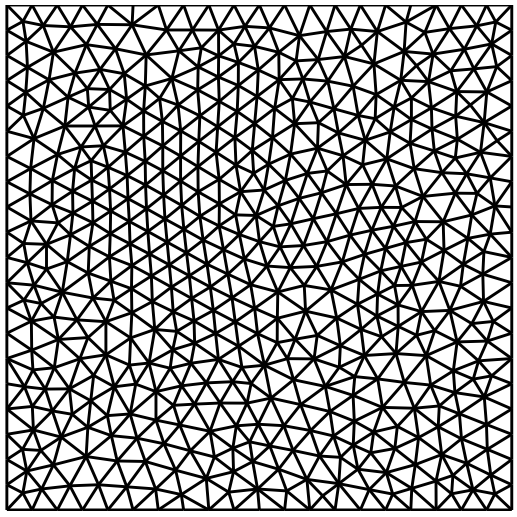
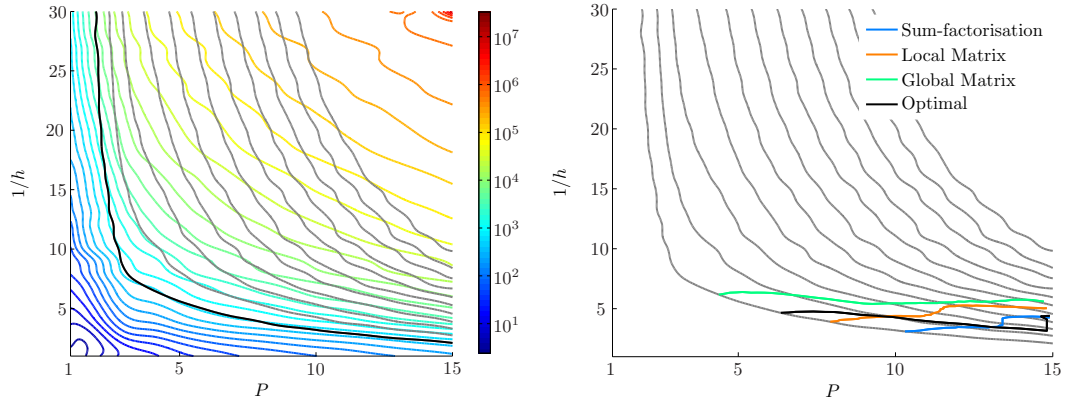


Figure 7.6: Unstructured triangular mesh with $h = 1/20$.

set of different *hp*-configurations. Although the trend is similar to the quadrilateral case, note that the data are not as smooth. This may be accounted to the fact that $1/h$ does not linearly scale with $\log_2(\mathcal{E})$ as in the quadrilateral case. However, this does not prevent a similar analysis to derive the path of minimal run-time as in



(a) Run-time (μs) and error contours $||\epsilon||_{L_2} = 10^i\,(1 \leq i \leq 11)$

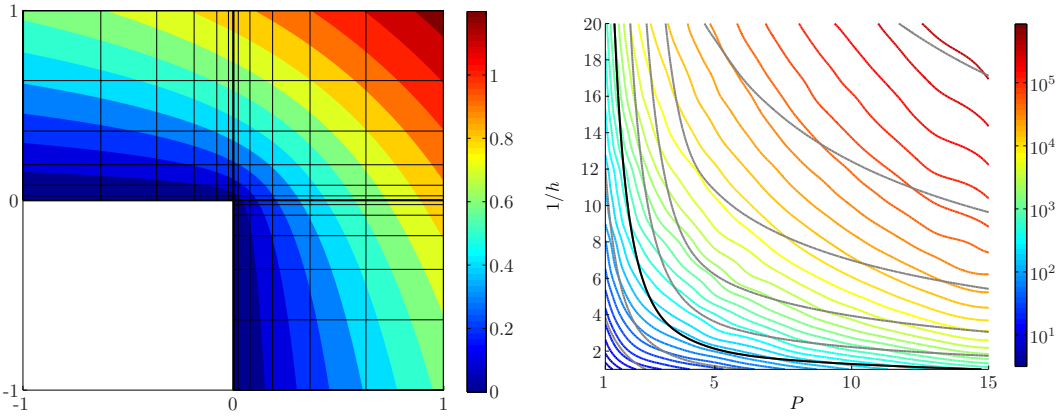
(b) Path of minimal run-time

Figure 7.7: Triangular spectral/*hp* discretisations to approximate the Helmholtz problem with smooth exact solution (7.5).

the previous examples. This result is summarised in Figure 7.7(b). We can observe that for a 10% error, again a sixth-order spectral/*hp* expansion with characteristic mesh-size $h = 1/5$ – which corresponds to 38 triangles – is the computationally most efficient *hp*-discretisation (see also Table 7.1). The run-time associated with this optimal triangular *hp*-discretisation appears to more than twice the run-time needed for the optimal quadrilateral expansion. As a result, for the uniform test-case under consideration, the quadrilateral sixth-order expansion employing 3×3 elements can be considered as the true optimal spectral/*hp* expansion. However, we would like to remark that this apparent superiority of the quadrilateral expansion is reinforced by the tensorial structure of the exact solution. Indeed, if we rotate the exact solution with 30°, the overhead of the triangular expansion is reduced with 20%. Regarding the convergence of the error in this triangular case, we can draw analogous conclusions as for the quadrilateral case.

7.5 Test problem 4: Quadrilateral spectral/*hp* discretisations for an irregular solution

Finally, we include an example which exact solution is not infinitely smooth. Therefore, we consider the L-shape domain problem shown in Figure 7.8. The forcing



(a) L-shape domain, radical mesh (with $|\mathcal{E}|^{1d} = 7$) and exact solution

(b) Run-time (μs) and error contours $||\epsilon||_{L_2} = 10^{-i}\,(2 \leq i \leq 11)$

Figure 7.8: Quadrilateral spectral/*hp* discretisations to approximate the Helmholtz problem with exact solution (7.11).

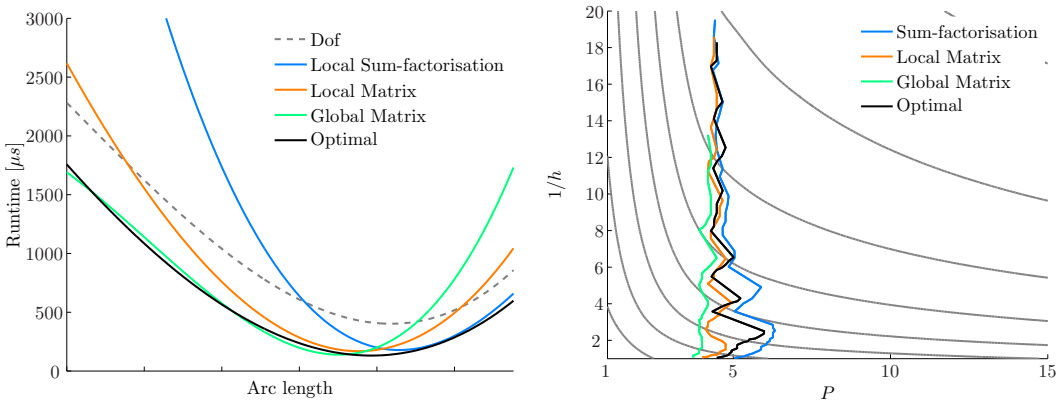
function f and Dirichlet boundary conditions are chosen such that the Helmholtz equation (7.1) yields the exact solution

$$u\left(x(r,\theta), y(r,\theta)\right) = r^{\frac{2}{3}} \sin(\frac{2}{3}\theta + \frac{\pi}{3}), \tag{7.11}$$

where (r, θ) are the traditional polar coordinates. Because of the term $r^{\frac{2}{3}}$, the gradient of the solution will exhibit a singularity at the re-entrant corner. This problem has been extensively studied in a FEM and *hp*-FEM context, often with an emphasis on adaptive refinement, see e.g. (Babuška & Suri 1994). In this study, we do not consider adaptivity but adopt a discretisation strategy where we account for the locality of the singularity by using a radical rather than equispaced mesh. The grid-points in each interval $[0, \pm 1]$ of a radical mesh are located at

$$x_i = \pm \left(\frac{i}{|\mathcal{E}|^{1d}} \right)^{\beta}, \quad i = 0, 1, \ldots, |\mathcal{E}|^{1d}, \tag{7.12}$$

where the parameter β is chosen as $\beta = 3$ (according to (Seshaiyer & Suri 2000)) and $|\mathcal{E}|^{1d}$ is the number of elements in one dimension. The error and cost contours are depicted in Figure 7.8(b). Note that the error contours follow a different pattern than for the cases with a smooth solution. This problem therefore has an entirely different path of minimal run-time, see Figure 7.9(b). There are various remarkable features in this figure. First it can be seen that for a given error tolerance, all



(a) Computational cost along 10^{-4} error contour

(b) Path of minimal run-time

Figure 7.9: Quadrilateral spectral/hp discretisations to approximate the Helmholtz problem with exact solution (7.11).

strategies appear to suggest almost the same (h, P) discretisation. This includes the discretisation which minimises the number of degrees-of-freedom rather than the run-time. This may be appreciated by considering Figure 7.9(a) which shows that the minimum – based upon DOFs – along the 10^{-4} error contour is now much sharper than for the smooth test-problems, see for example Figure 7.2(a). From Figure 7.9(b) it appears that the effect of the different implementation strategies do not cause significantly different run-time to overcome the overhead of additional DOFs. As a result, all minima seem to coincide around the same discretisation. Secondly, it can be observed that the path of minimal run-time converges along the h-direction, rather than the P-direction. This is plausible as the spectral/hp element method does not exhibit exponential error-convergence with respect to the polynomial order because of the singularity in the solution present at the corner of the domain. In addition the use of a radical mesh, which has been shown to be optimal for h-type FEM in this context, and so may contribute to this observation. Finally, Figure 7.9(b) clearly indicates that a fifth-order expansion is now optimal and that the mesh-size should be varied according to the desired accuracy. Although this type of h-type convergence for non-smooth problems is typically associated to low-order finite element methods ($P = 1$, or sometimes $1 \leq P \leq 3$), the resulting fifth-order optimum indicates that high-order finite element methods also have their

use in solving singular problems.

Chapter 8

Conclusions

8.1 Summary

This thesis considered optimising the implementation of the spectral/*hp* element method. We therefore have studied some of the aspects that may contribute to this goal. All these aspects can be related to the following issues, which we have identified as two of the major challenges that arise in developing an efficient implementation of the spectral/*hp* element method:

- implementing the mathematical structure of the technique in a digestible, generic and coherent manner, and
- designing and implementing the numerical methods and data structures in a matter so that both high- and low-order discretisations can be efficiently applied.

We have seen in the introduction that when applied to the implementation of the advection-diffusion equation, both objectives above have lead to the following five questions below. In this chapter, we want to conclude the thesis by formulating an answer to these questions, thereby summarising each of the previous chapters of this thesis.

How to encapsulate the fundamental concepts related to the spectral/*hp* element method? We have started by describing how the mathematical construction of a spectral/*hp* discretisation can be emulated in an object-oriented environment. We have demonstrated how we have chosen to encapsulate the spectral/*hp* element discretisation in a collection of carefully designed libraries and classes, resulting in a generic, flexible and modular software library that allows user to implement their own spectral/*hp* solvers. We have also given a list of good coding practices that may help to enhance the efficiency of an algorithm.

How to apply the time-stepping in a generic and efficient way? We proposed a generic framework, both in terms of algorithms and implementations, that facilitates the application of a broad range of time-stepping schemes in a uniform way. We based our algorithm on Butcher's unifying theory of General Linear Methods, a concept widely accepted within the ODE community but little known from a PDE point of view. The framework should allow CFD users, who often tend to limit themselves to a single (family of) schemes, to explore the plethora of different methods that exist – implicit versus explicit, multi-stage versus multi-step – without any additional effort. We illustrated that the abstract character of the framework allows for an object-oriented implementation where switching between different schemes is as simple as changing an input parameter. Although we first presented an generic ODE solving framework, the main emphasis of this work is on time-integrating PDEs. Therefore, we first showed how IMEX schemes – a family of time-stepping schemes popular within the CFD community – can be formulated as a General Linear Method. Then we demonstrated how through some modifications, the framework can be adapted to accommodate the time-integration of PDEs in a generic and computationally efficient way. Overall we believe this study provides the essential building blocks for time-integrating PDEs in a unified way. Finally, please note that even though we mainly followed a finite element procedure for the spatial discretisation the presented techniques are believed to be general and can be used within a finite volume or finite difference context.

How to optimise the evaluation of the spectral/*hp* element operators We have shown that in order to implement the spectral/hp element method for a broad range of polynomial orders ($1 \leq P \leq 15$), a spectral/hp element code should ideally support three different implementation strategies to evaluate the finite element operators. This allows a hybrid strategy based upon the polynomial order. For low-order expansions, as is common practice we have shown that the evaluation using global matrices is most efficient while for high-order expansions, one should preferably employ the sum-factorisation technique. If operating in the intermediate regime between low and high-order, the evaluation using local matrices is often the most efficient option. Furthermore, we demonstrated that the break-even points between these different polynomial regimes depend on the operator to be evaluated, the shape of the element and the computer on which the code is run. We have presented both theoretical estimates as well as computational test confirming this behaviour for different two-dimensional finite element operators.

How to optimise the solution of the linear system? We have acknowledged that there are two distinct ways of applying the idea of multi-level static condensation or substructuring in order to solve linear systems. Both the bottom-up and the top-down approaches are based upon a nested bisection algorithm to reorder the degrees of freedom. A theoretical analysis clearly showed that the bottom-up approach is preferred above the top down-approach. A computational comparison has also shown that for high-order expansions, adopting the (bottom-up) multi-level static condensation technique leads to a significant reduction in run-time when compared to the more traditional approach of a bandwidth minimised single-level static condensation technique. This reduction factor may be as high as a factor six. Only for linear finite elements, the technique of bandwidth minimisation prevails. We have also demonstrated that the idea of multi-level static condensation is fully compatible with Dirichlet boundary conditions.

What is the optimal *hp*-discretisation at which to run your code? In the Chapter 7, we have investigated how to select the parameters (h, P) of a spectral/hp

discretisations in order to minimise the run-time when solving an elliptic problem up to predefined accuracy. As expected, the numerical results indicate that in case of smooth solutions, one should fix the mesh and vary the polynomial order according to the desired accuracy (p-convergence). In addition, our computational investigation has however highlighted the not quite so intuitive result that for a low error level of 10% a reasonably coarse mesh with a sixth-order spectral/hp expansions minimised the run-time. For non-smooth solutions on the other hand, and consistent with theory, we observed that the run-time can be minimised by fixing the polynomial order of the expansion and refining the mesh according to the desired error tolerance (h-convergence). However, for the non-smooth test problem under consideration, we observed that a polynomial order of as high as $P = 5$ was optimal, thereby promoting the use of high-order expansions for problems with corner-type singularities, at least when using a radical mesh distribution.

8.2 Recommendations

The results from this thesis should provide any spectral/hp element code developer a fundamental insight into various aspects that may contribute to an efficient implementation of this high-order method. We put particular emphasis on designing algorithms that allow the user to go *from* h *to* p *efficiently*, that is, algorithms which are both efficient for low-order methods (h-type finite elements) and high-order methods (p-type finite element methods). This can be accomplished by accommodating different implementation strategies depending on the polynomial order of the expansion.

For the scope of this work, we did only consider direct solution methods to solve the linear systems that arise in the spectral/hp element method. However, some of the results can also be thought to be relevant for iterative solution methods such as the conjugate gradient method. As this iterative method is based upon the action of the operator rather than on its inverse, the observed performance differences of the various implementation strategies, see Chapter 5, can certainly be exploited to enhance the efficiency of iterative solution techniques. Moreover, adopting an

iterative solution method is also assumed to lead to different results for the analysis in Chapter 7, where we searched for the (h, P) discretisation that minimises the run-time for a predefined level of accuracy. This can be appreciated considering the fact that the required run-time to solve the linear system due to a certain (h, P) discretisation will then also depend on the implementation strategy. Something which is not the case for the direct solution strategy we adopted and which does not depend on the implementation strategy. One can therefore expect that adopting a iterative solution for such an analysis will lead to more pronounced differences in the results. In addition, the multi-level static condensation technique may be of use for preconditioning the conjugate gradient method.

As we have restricted ourselves in this thesis to two-dimensional expansions only, it would also be interesting to perform a similar analysis for three-dimensional expansions. Not only is the advantage of the sum-factorisation known to be stronger in three dimensions, this can also be used as an opportunity to explore these strategies in the context of parallel computing. There, other factors such as memory considerations and again iterative solution techniques will come into play. It is expected that the techniques which require less memory such as the matrix-free sum-factorisation technique, may have an advantage over the other methods. Memory intensive techniques such as the global matrix approach may even be too expensive as the required global matrices may become to big to store. As already mentioned in Chapter 6, the presented multi-level static condensation technique can be readily applied in 3D. The underlying implementation is based upon a matrix reordering algorithm and as such, it does not distinguish between a matrix due to a two-dimensional or a three-dimensional problem. However, we have also indicated in Chapter 6 that the technique may be less efficient in 3D than in 2D. This can be appreciated by the fact that the matrix reordering uses a nested bisection algorithm and consequently, it effectively reduces a 2D problem to a 1D problem and a 3D problem to a 2D one. Relatively, this reduction is more favourable for the 2D than for the 3D case.

The ingredients presented in this method should also facilitate other studies that try to answer one of the most pertinent questions related to the application of the spectral/*hp* element method or any other high-order finite element method:

what is the best mesh-resolution and expansion order to run our simulations? We believe this choice currently is largely based on tradition, intuition or experience rather than on theoretically founded arguments. The principles presented in this thesis should provide a basis for performing such comparative studies. We have given a first attempt in Chapter 7 by investigating this for the simple example of the steady Helmholtz equation. An interesting extension would be to repeat this study for a time-dependent partial-differential equation such as the advection-diffusion equation. This time-dependency will, next to the parameters (h, P) due to the spatial discretisation, introduce two more parameters due to the temporal discretisation, the time-step Δt and the order of the time-stepping scheme. To further complicate things, both parameter pairs are known to depend on each other.

Bibliography

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., & Sorensen, D. (1999). *LAPACK Users' Guide* (Third ed.). Philadelphia, PA: Society for Industrial and Applied Mathematics. `http://www.netlib.org/lapack`.

Ascher, U. M., Ruuth, S. J., & Spiteri, R. J. (1997). Implicit–explicit Runge–Kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics: Transactions of IMACS 25*(2–3), 151–167.

Ascher, U. M., Ruuth, S. J., & Wetton, B. T. R. (1995). Implicit-explicit methods for time-dependent partial differential equations. *SIAM J. Numer. Anal. 32*(3), 797–823.

Babuška, I. & Suri, M. (1994). The p and hp versions of the finite element method, basic principles and properties. *SIAM Rev. 36*(4), 578–632.

Bagheri, B., Scott, L. R., & Zhang, S. (1994). Implementing and using high-order finite element methods. *Finite Elem. Anal. Des. 16*(3-4), 175–189.

Bernard, P.-E., Deleersnijder, E., Legat, V., & Remacle, J.-F. (2008). Dispersion analysis of discontinuous Galerkin schemes applied to Poincar, Kelvin and Rossby waves. *J. Sci. Comput. 34*, 26–47.

Bernard, P.-E., Remacle, J.-F., Combien, R., Legat, V., & Hillewaert, K. (2009). High order discontinous Galerkin schemes on general 2D manifolds applied to the shallow water equations. *J. Comp. Phys. 228*, 6514–6535.

Burrage, K. & Butcher, J. C. (1980). Non-linear stability of a general class of differential equation methods. *BIT 20*, 185–203.

Butcher, J. C. (1987). *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley, Chichester.

Butcher, J. C. (1997). An introduction to almost Runge-Kutta methods. *Appl. Numer. Math. 24*, 331–342.

Butcher, J. C. (2006). General linear methods. *Acta Numerica 15*, 157–256.

Deville, M. O., Fischer, P. F., & Mund, E. H. (2002). *High-order methods for incompressible fluid flow*. Cambridge monographs on applied and computational mathematics. Cambridge: Cambridge University Press.

Donea, J., Giuliani, S., Laval, H., & Quartapelle, L. (1984). Time-accurate solution of advection-diffusion problems by finite elements. *Comput. MEthods Appl. Mech. Engrg. 45*(1-3), 123–145.

Donelson, J. & Hansen, E. (1971). Cyclic composite multistep predictor-corrector methods. *Siam J. Numer. Anal. 8*(1), 137–157.

Dongarra, J., Du Croz, J., Hammarling, S., Hanson, R., & Duff, I. (1988). Blas: The Basis Linear Algebra Subprograms. `http://www.netlib.org/blas`.

Dubiner, M. (1991). Spectral methods on triangles and other domains. *J. Sci. Comput. 6*(4), 345–390.

Elssel, K. & Voss, H. (2008). Reducing sparse nonlinear eigenproblems by automated multi-level substructuring. *Advances in Engineering Software 39*, 828–838.

Eskilsson, C. & Sherwin, S. J. (2004). A triangular spectral/hp discontinuous Galerkin method for modelling 2D shallow water equations. *Int. J. Numer. Meth. Fluids 45*, 605–623.

Fischer, P. & Gottlieb, D. (1996). On the optimal number of subdomains for hyperbolic problems on parallel computers. *Int. J. Supercomputer Appl. High Perform. Comput. 11*, 65–76.

Gao, W., Xiaoye, S., Chao, Y., & Zhaojun, B. (2008). An implementation and evaluation of the AMLS method for sparse eigenvalue problems. *ACM Trans. Math. Softw. 34*(4), 20:1–20:28.

Gottlieb, D. & Orszag, S. A. (1977). *Numerical analysis of spectral methods: theory and applications*. CBMS-NSF. Philadelphia: Society for Industrial and Applied Mathematics.

Hesthaven, J. & Warburton, T. (2002). Nodal high-order methods on unstructured grids: I. Time-domain solution of Maxwell's equations. *J. Comp. Phys. 181*, 186–221.

Hesthaven, J. S. (1997). A Stable Penalty Method for the Compressible Navier–Stokes Equations: II. One-Dimensional Domain Decomposition Schemes. *SIAM J. Sci. Comput. 18*(3), 658–685.

Hesthaven, J. S. (1998). A Stable Penalty Method for the Compressible Navier–Stokes Equations: III. Multidimensional Domain Decomposition Schemes. *SIAM J. Sci. Comput. 20*(1), 62–93.

Hesthaven, J. S. & Warburton, T. (2008). *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Texts in Applied Mathematics 54. New York: Springer Verlag.

Hughes, T., Cottrell, J., & Bazilevs, Y. (2005). Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comp. Meth. Appl. Mech. Engrg. 194*(39-41), 4135–4195.

Hughes, T. J. R. (1987). *The finite element method*. Prentis Hall, New Jersey.

Karniadakis, G. E., Israeli, M., & Orszag, S. A. (1991). High-order splitting methods for the incompressible Navier-Stokes equations. *J. Comput. Phys. 97*, 414–443.

Karniadakis, G. E. & Sherwin, S. J. (2005). *Spectral/hp element methods for computational fluid dynamics* (second ed.). Numer. Math. Sci. Comp. Oxford: Oxford University Press.

Karypis, G. & Kumar, V. (1998). Metis. A software package for partitioning unstructered graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices. Version 4.0. http://www.cs.umn.edu/~karypis.

Kirby, R. & Karniadakis, G. (2003). De-aliasing on non-uniform grids: algorithms and applications. *J. Comput. Phys. 191*(1), 249–264.

Kirby, R. & Sherwin, S. (2006). The Nektar++ project. http://www.nektar.info.

Kirby, R. C., Knepley, M., & Scott, L. R. (2004). Evaluation of the action of finite element operators. Technical Report TR-2004-07, University of Chicago.

Noye, B. J. & Tan, H. H. (1989). Finite difference methods for solving the two-dimensional advection-diffusion equation. *Int. J. Numer. Meth. Fluids 9*(1), 75–89.

OpenFOAM (2010). The Open Source CFD Toolbox. http://www.openfoam.org.

Orszag, S. A. (1980). Spectral methods for problems in complex geometries. *J. Comput. Phys. 37*(1), 70–92.

Patera, A. T. (1984). A spectral element method for fluid dynamics: Laminar flow in a channel expansion. *J. Comput. Phys. 54*, 468–488.

Rachowicz, W. & Zdunek, A. (2009). Automated multi-level substructering (AMLS) for electromagnetics. *Comp. Meth. Appl. Mech. Engrg. 1998*, 1224–1234.

Rattenbury, N. (2005). *Almost Runge-Kutta methods for stiff and non-stiff problems*. Ph. D. thesis, The University of Auckland.

Remacle, J. F., Flaherty, J. E., & Shepard, M. S. (2003). An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. *SIAM Rev. 45*, 55–73.

Remington, K. & Pozo, R. (1996). The NIST Sparse Blas: Basic Toolkit v.0.9b. http://math.nist.gov/spblas/original.html.

Rønquist, E. M. (1988). *Optimal spectral element methods for unsteady three-dimensional incompressible Navier-Stokes equations.* Ph. D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Ruuth, S. J. & Spiteri, R. J. (2004). High-order strong-stability-preserving Runge-Kutta methods with downwind-biased spatial discretizations. *Siam J. Numer. Anal. 42*(3), 974–996.

Schiesser, W. E. (1991). *The numerical method of lines: integration of partial differential equations.* Academic Press, San Diego.

Seshaiyer, P. & Suri, M. (2000). *hp* submeshing via non-conforming finite element methods. *Comp. Meth. Appl. Mech. Engrg. 189*(3), 1011–1030.

Sherwin, S. & Karniadakis, G. (1996). Tetrahedral hp finite elements: Algorithms and flow simulations. *J. Comp. Phys 124*, 14–45.

Shu, C.-W. (1987). TVD time discretizations. *SIAM J. Sci. Stat. Comput. 9*, 1073–1084.

Smith, B., Bjorstad, P., & Gropp, W. (1996). *Domain decomposition. Parallel multi-level methods for elliptic differntial equations.* Cambridge University Press.

Song, C. & Yuan, M. (1990). Simulation of vortex-shedding flow about a circular cylinder at high Reynolds numbers. *J. Fluids Eng. 112.*

Souza Carmo, B. (2009). *On Wake Interference in the flow around two circular cylinders: Direct stability analysis and flow-induced vibrations.* Ph. D. thesis, Department of Aeronautics, Imperial College London.

Stroustrup, B. (2000). *The C++ Programming Language.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Szabó, B. & Babuška, I. (1991). *Finite Element Analysis.* New York: Wiley.

Wasberg, C. E. & Gottlieb, D. (2000). Optimal Decomposition of the Domain in Spectral Methods for Wave-Like Phenomena. *SIAM J. Sci. Comput. 22*(2), 617–632.

Whaley, R., Petitet, A., & Dongarra, J. (2001). Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing 27*(1–2), 3–35. `http://math-atlas.sourcefourge.net`.

Zienkiewicz, O. C. & Taylor, R. L. (1989). *The finite element method* (Fourth Edition ed.). New York: McGraw-Hill.

Appendix A

Theoretical operation count

A.1 Notation and assumptions

A.1.1 Definitions

With regard to a two-dimensional spectral/*hp* expansion, we define

- P_1 as the expansion order of the basis in direction 1,
- P_2 as the expansion order of the basis in direction 2,
- $N_1 = P_1 + 1$ as the number of modes of the basis in direction 1,
- $N_2 = P_2 + 2$ as the number of modes of the basis in direction 2,
- N_{tot} as the total number of modes of the expansion,
- Q_1 as the number of quadrature points in direction 1,
- Q_2 as the number of quadrature points in direction 2, and
- Q_{tot} as the total number of quadrature points on the element.

For a quadrilateral expansion, the total number of modes is equal to $N_{tot} = N_1 N_2$ while for a triangular expansion we assume $N_1 \leq N_2$ such that $N_{tot} = \frac{1}{2}N_1(N_1+1) + N_1(N_2 - N_1)$. In both cases, we use a tensorial quadrature rule with $Q_{tot} = Q_1 Q_2$.

A.1.2 Matrix operators

Table A.1.2 gives an overview of all the matrix operators that will be used in this appendix.

A.1.3 Operation count of matrix operators

For

- a dense matrix $\boldsymbol{A}$ of size $m \times n$,
- a dense matrix $\boldsymbol{B}$ of size $n \times k$, and
- a vector $\boldsymbol{b}$ of size n,

we will assume the following operation counts:

	Operation Count		
	multiplications	additions	total
matrix-vector product $\boldsymbol{Ax}$	mn	mn	$2mn$
matrix-matrix product $\boldsymbol{AX}$	mnk	mnk	$2mnk$

Note that it in principle is possible to evaluate the matrix-vector multiplication using only $m(n-1)$ additions. However, for simplicity we will follow the operation count displayed in the table above.

Furthermore, for (block-)diagonal matrices, we only count the floating point operations due to the diagonal entries.

A.2 The sum-factorisation approach

We will first determine the operation count required for the factorised evaluation of the two core operators, i.e. the operator $\boldsymbol{B}$ and its transpose $\boldsymbol{B}^\top$. This will facilitate the operation count for the finite element operators afterwards.

A.2.1 The operator $\boldsymbol{B}$ and its transpose operator $\boldsymbol{B}^\top$

Quadrilateral expansions According to Section 5.1.1, the operation

$$\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}, \tag{A.1}$$

matrix	type	dimension	definition	name
$\boldsymbol{B}$	dense	$Q_{tot} \times N_{tot}$	$\boldsymbol{B}[i][j] = \phi_j(\boldsymbol{\xi}_i)$	discrete 2D basis
$\boldsymbol{B}_m^a$	dense	$Q_m \times N_m$	$\boldsymbol{B}_m^a[i][j] = \psi_j^a(\xi_i)$	discrete 1D basis in direction m
$\boldsymbol{B}_p^b$	dense	$Q_2 \times f(p)$	$\boldsymbol{B}_p^b[i][j] = \phi_{pj}^b(\xi_i)$	generalised 1D basis
$\boldsymbol{D}_m^{1d}$	dense	$Q_m \times Q_m$	$\boldsymbol{D}_m^{1d}[i][j] = \frac{dh_j(\xi_i)}{d\xi}$	1D derivative matrix in direction m
$\boldsymbol{M}$	dense	$N_{tot} \times N_{tot}$	$\boldsymbol{M}[i][j] = (\phi_i, \phi_j)$	mass matrix
$\boldsymbol{H}$	dense	$N_{tot} \times N_{tot}$	$\boldsymbol{H}[i][j] = (\phi_i, \phi_j) + \lambda\,(\nabla\phi_i, \nabla\phi_j)$	Helmholtz matrix
$\bar{\boldsymbol{B}}$	dense	$Q_{tot} \times N_{tot}$	$\bar{\boldsymbol{B}}[i][j] = \phi_j(\boldsymbol{\xi}_i)\omega_i\,\lvert J(\boldsymbol{\xi}_i)\rvert$	weighted discrete 2D basis
$\boldsymbol{W}$	diagonal	$Q_{tot} \times Q_{tot}$	$\boldsymbol{W}[i][i] = \omega_i\,\lvert J(\boldsymbol{\xi}_i)\rvert$	quadrature metric
$\boldsymbol{G}^{mn}$	diagonal	$Q_{tot} \times Q_{tot}$	$\boldsymbol{G}^{mn}[i][i] = \omega_i\,\lvert J(\boldsymbol{\xi}_i)\rvert \sum_{j=\{1,2\}} \prod_{k=\{m,n\}} d_{jk}(\boldsymbol{\xi}_i)$	Laplacian metric
$\underline{\boldsymbol{B}}^b$	block-diagonal	$N_1 \times N_1$	$\underline{\boldsymbol{B}}^b[i][i] = \boldsymbol{B}_p^b$	generalised 1D basis
$\boldsymbol{D}_1$	sparse	$Q_{tot} \times Q_{tot}$	$\boldsymbol{D}_0 = \boldsymbol{I} \otimes \boldsymbol{D}_1^{1d}$	derivative matrix in direction 1
$\boldsymbol{D}_2$	sparse	$Q_{tot} \times Q_{tot}$	$\boldsymbol{D}_0 = \boldsymbol{D}_2^{1d} \otimes \boldsymbol{I}$	derivative matrix in direction 2

Table A.1: Overview of the matrix operators used in this section. Note that $d_{jk}(\boldsymbol{\xi})$ is the derivative metric respectively defined as $\frac{\partial \xi_k}{\partial x_j}$ and $\frac{\partial \eta_k}{\partial x_j}$ for quadrilateral and triangular elements.

can be evaluated as

$$\boldsymbol{U} = \boldsymbol{B}_1^a \hat{\boldsymbol{U}} \boldsymbol{B}_2^{a\top}. \tag{A.2}$$

This yields the operation count:

		Operation Count	
		multiplications	additions
step 1	$\boldsymbol{V} = \hat{\boldsymbol{U}} \boldsymbol{B}_2^{a\top}$	$N_{tot} Q_2$	$N_{tot} Q_2$
step 2	$\boldsymbol{U} = \boldsymbol{B}_1^a \boldsymbol{V}$	$N_1 Q_{tot}$	$N_1 Q_{tot}$
total		$N_{tot} Q_2 + N_1 Q_{tot}$	$N_{tot} Q_2 + N_1 Q_{tot}$
		$2(N_{tot} Q_2 + N_1 Q_{tot})$	

Analogously, the transpose operator

$$\hat{\boldsymbol{u}} = \boldsymbol{B}^\top \boldsymbol{u}, \tag{A.3}$$

can be evaluated in an equal number of floating point operations:

		Operation Count	
		multiplications	additions
step 1	$\boldsymbol{V} = \boldsymbol{B}_1^{a\top} \boldsymbol{U}$	$N_1 Q_{tot}$	$N_1 Q_{tot}$
step 2	$\hat{\boldsymbol{U}} = \boldsymbol{V} \boldsymbol{B}_2^a$	$N_{tot} Q_2$	$N_{tot} Q_2$
total		$N_{tot} Q_2 + N_1 Q_{tot}$	$N_{tot} Q_2 + N_1 Q_{tot}$
		$2(N_{tot} Q_2 + N_1 Q_{tot})$	

Triangular expansions For triangular expansions, it has been shown in Section 5.1.1 that (A.1) can be evaluated as

$$\boldsymbol{u} = (\boldsymbol{B}_1^a \otimes \boldsymbol{I})\, \underline{\boldsymbol{B}}^b \hat{\boldsymbol{u}}, \tag{A.4}$$

This requires the following number of floating point operations:

		Operation Count	
		multiplications	additions
step 1	$\boldsymbol{v} = \underline{\boldsymbol{B}}^b \hat{\boldsymbol{u}}$	$N_{tot} Q_2$	$N_{tot} Q_2$
step 2	*degenerate vertex*	Q_2	Q_2
step 3	$\boldsymbol{U} = \boldsymbol{B}_1^a \boldsymbol{V}^\top$	$N_1 Q_{tot}$	$N_1 Q_{tot}$
total		$N_{tot} Q_2 + N_1 Q_{tot} + Q_2$	$N_{tot} Q_2 + N_1 Q_{tot} + Q_2$
		$2(N_{tot} Q_2 + N_1 Q_{tot} + Q_2)$	

The second step is required to properly take into account the degenerate vertex mode of the C^0 continuous modal expansion, see (Karniadakis & Sherwin 2005).

Analogously, for the transpose operator (A.3) we can derive:

		Operation Count	
		multiplications	additions
step 1	$\boldsymbol{V} = \boldsymbol{U}^\top \boldsymbol{B}_1^a$	$N_1 Q_{tot}$	$N_1 Q_{tot}$
step 2	*degenerate vertex*	Q_2	Q_2
step 3	$\hat{\boldsymbol{u}} = \underline{\boldsymbol{B}}^b \boldsymbol{v}$	$N_{tot} Q_2$	$N_{tot} Q_2$
total		$N_{tot}Q_2 + N_1 Q_{tot} + Q_2$	$N_{tot}Q_2 + N_1 Q_{tot} + Q_2$
		$2(N_{tot}Q_2 + N_1 Q_{tot} + Q_2)$	

A.2.2 Backward transformation

Evaluated as:

$$\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}. \tag{A.5}$$

Operation count:

	Operation Count	
	multiplications	additions
total	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$
	$2(N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2)$	

where $\gamma = 0$ for quadrilateral elements and $\gamma = 1$ for triangular elements.

A.2.3 Inner product

Evaluated as:

$$\hat{\boldsymbol{u}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{u}. \tag{A.6}$$

Operation count:

		Operation Count	
		multiplications	additions
step 1	$\boldsymbol{v} = \boldsymbol{W}\boldsymbol{u}$	Q_{tot}	
step 2	$\hat{\boldsymbol{u}} = \boldsymbol{B}^\top \boldsymbol{v}$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$
total		$N_{tot}Q_2 + (N_1+1) Q_{tot} + \gamma Q_2$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$
		$2(N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2) + Q_{tot}$	

where $\gamma = 0$ for quadrilateral elements and $\gamma = 1$ for triangular elements.

A.2.4 Mass matrix operator

Evaluated as:

$$\hat{\boldsymbol{y}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{B} \hat{\boldsymbol{u}}. \tag{A.7}$$

Operation count:

		Operation Count	
		multiplications	additions
step 1	$\boldsymbol{v}_1 = \boldsymbol{B}\hat{\boldsymbol{u}}$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$
step 2	$\boldsymbol{v}_2 = \boldsymbol{W}\boldsymbol{v}_1$	Q_{tot}	
step 3	$\hat{\boldsymbol{y}} = \boldsymbol{B}^\top \boldsymbol{v}_2$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$
total		$2(N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2) + Q_{tot}$	$2(N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2)$
		$4(N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2) + Q_{tot}$	

where $\gamma = 0$ for quadrilateral elements and $\gamma = 1$ for triangular elements.

A.2.5 Helmholtz operator

Evaluated as

$$\hat{\boldsymbol{y}} = \boldsymbol{B}^\top \left(\begin{bmatrix} \boldsymbol{D}_{\xi_1}^\top & \boldsymbol{D}_{\xi_2}\top \end{bmatrix} \begin{bmatrix} \boldsymbol{G}^{11} & \boldsymbol{G}^{12} \\ \boldsymbol{G}^{21} & \boldsymbol{G}^{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_{\xi_1} \\ \boldsymbol{D}_{\xi_2} \end{bmatrix} + \boldsymbol{W} \right) \boldsymbol{B}\hat{\boldsymbol{u}} \tag{A.8}$$

Operation count:

		Operation Count	
		multiplications	additions
step 1	$\boldsymbol{v}_1 = \boldsymbol{B}\hat{\boldsymbol{u}}$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$
step 2	$\boldsymbol{v}_2 = \boldsymbol{W}\boldsymbol{v}_1$	Q_{tot}	
step 3	$\boldsymbol{V}_3 = \boldsymbol{D}_1^{1d}\boldsymbol{V}_1$	$Q_{tot}Q_1$	$Q_{tot}Q_1$
step 4	$\boldsymbol{V}_4 = \boldsymbol{V}_1(\boldsymbol{D}_2^{1d})^\top$	$Q_{tot}Q_2$	$Q_{tot}Q_2$
step 5	$\boldsymbol{v}_5 = \boldsymbol{G}^{11}\boldsymbol{v}_3 + \boldsymbol{G}^{12}\boldsymbol{v}_4$	$2Q_{tot}$	Q_{tot}
step 6	$\boldsymbol{v}_6 = \boldsymbol{G}^{21}\boldsymbol{v}_3 + \boldsymbol{G}^{22}\boldsymbol{v}_4$	$2Q_{tot}$	Q_{tot}
step 7	$\boldsymbol{V}_7 = (\boldsymbol{D}_1^{1d})^\top \boldsymbol{V}_5$	$Q_{tot}Q_1$	$Q_{tot}Q_1$
step 8	$\boldsymbol{V}_8 = \boldsymbol{V}_6\boldsymbol{D}_2^{1d}$	$Q_{tot}Q_2$	$Q_{tot}Q_2$
step 9	$\boldsymbol{v}_9 = \boldsymbol{v}_1 + \boldsymbol{v}_7 + \boldsymbol{v}_8$		$2Q_{tot}$
step 10	$\hat{\boldsymbol{y}} = \boldsymbol{B}^\top \boldsymbol{v}_9$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$	$N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2$
total		$2(N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2) + Q_{tot}(2Q_1 + 2Q_2 + 5)$	$2(N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2) + Q_{tot}(2Q_1 + 2Q_2 + 4)$
		$4(N_{tot}Q_2 + N_1 Q_{tot} + \gamma Q_2) + Q_{tot}(4Q_1 + 4Q_2 + 9)$	

where $\gamma = 0$ for quadrilateral elements and $\gamma = 1$ for triangular elements.

A.3 The local-matrix approach

A.3.1 Backward transformation

Evaluated as:

$$\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}. \tag{A.9}$$

Operation count:

	Operation Count	
	multiplications	additions
total	$N_{tot}Q_{tot}$	$N_{tot}Q_{tot}$
	$2(N_{tot}Q_{tot})$	

A.3.2 Inner product

Evaluated as:

$$\hat{\boldsymbol{u}} = \bar{\boldsymbol{B}}^{\top}\boldsymbol{u}. \tag{A.10}$$

Operation count:

	Operation Count	
	multiplications	additions
total	$N_{tot}Q_{tot}$	$N_{tot}Q_{tot}$
	$2(N_{tot}Q_{tot})$	

A.3.3 Mass matrix operator

Evaluated as:

$$\hat{\boldsymbol{y}} = \boldsymbol{M}\hat{\boldsymbol{u}}. \tag{A.11}$$

Operation count:

	Operation Count	
	multiplications	additions
total	$N_{tot}^2 + \gamma Q_2$	N_{tot}^2
	$2N_{tot}^2$	

A.3.4 Helmholtz operator

Evaluated as:

$$\hat{\boldsymbol{y}} = \boldsymbol{H}\hat{\boldsymbol{u}}. \tag{A.12}$$

Operation count:

	Operation Count	
	multiplications	additions
total	$N_{tot}^2 + \gamma Q_2$	N_{tot}^2
	$2N_{tot}^2$	

A.4 A standard uniform case

Consider a uniform spectral/*hp* expansion such that $N = P + 1 = N_1 = N_2$ with the following Gaussian quadrature rule:

- Quadrilateral expansion
  - $Q = Q_1 = Q_2 = N + 1$ (Gauss-Lobatto-Legendre points).
- Triangular expansion
  - $Q_1 = N + 1$ (Gauss-Lobatto-Legendre points), and
  - $Q_2 = N$ (Gauss-Radau-Legendre points).

We have chosen the minimal amount of quadrature points needed to exactly integrate the linear operators, as e.g. explained in (Karniadakis & Sherwin 2005). This standard case yields the following operation count:

		sum-factorisation	local matrix
Quad	backward transformation	$4N^3 + 6N^2 + 2N$ $4P^3 + 18P^2 + 26P + 12$	$2N^4 + 4N^3 + 2N^2$ $2P^4 + 12P^3 + 26P^2 + 24P + 8$
	inner product	$4N^3 + 7N^2 + 4N + 1$ $4P^3 + 19P^2 + 30P + 16$	$2N^4 + 4N^3 + 2N^2$ $2P^4 + 12P^3 + 26P^2 + 24P + 8$
	mass matrix operator	$8N^3 + 13N^2 + 6N + 1$ $8P^3 + 37P^2 + 56P + 28$	$2N^4$ $2P^4 + 8P^3 + 12P^2 + 8P + 2$
	Helmholtz operator	$16N^3 + 45N^2 + 46N + 17$ $16P^3 + 93P^2 + 184P + 124$	$2N^4$ $2P^4 + 8P^3 + 12P^2 + 8P + 2$
Tri	backward transformation	$3N^3 + 3N^2 + 2N$ $3P^3 + 12P^2 + 17P + 8$	$N^4 + 2N^3 + N^2$ $P^4 + 6P^3 + 13P^2 + 12P + 4$
	inner product	$3N^3 + 4N^2 + 3N$ $3P^3 + 13P^2 + 20P + 10$	$N^4 + 2N^3 + N^2$ $P^4 + 6P^3 + 13P^2 + 12P + 4$
	mass matrix operator	$6N^3 + 7N^2 + 5N$ $6P^3 + 25P^2 + 37P + 18$	$0.5N^4 + N^3 + 0.5N^2$ $0.5P^4 + 3P^3 + 6.5P^2 + 6P + 2$
	Helmholtz operator	$14N^3 + 27N^2 + 17N$ $14P^3 + 69P^2 + 113P + 58$	$0.5N^4 + N^3 + 0.5N^2$ $0.5P^4 + 3P^3 + 6.5P^2 + 6P + 2$

A.5 The global matrix approach for a structured quadrilateral mesh

Consider a structured quadrilateral finite element mesh of $|\mathcal{E}^{1d}| \times |\mathcal{E}^{1d}| = |\mathcal{E}|$ elements. The modes of a P^{th}-order uniform spectral/hp expansion can then be classified as depicted in Figure A.1. This classification helps us to determine the

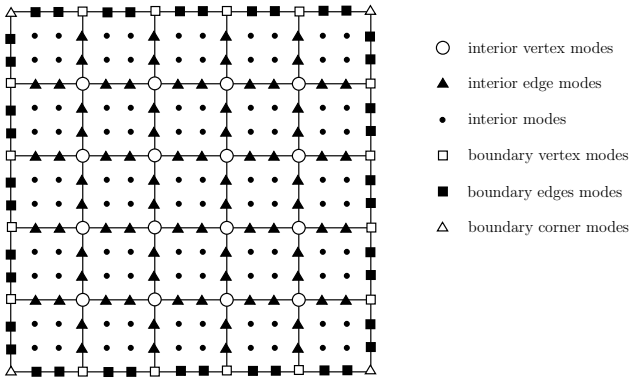


Figure A.1: Possible classification of the modes (or nodes) of a 3^{rd}-order uniform spectral/hp expansion on a structured 5×5 quadrilateral mesh.

number of non-zero entries in the global finite element operator $\boldsymbol{A}$, defined as

$$\boldsymbol{A}[i][j] = a(\Phi_i, \Phi_j). \tag{A.13}$$

where $a(\cdot,\cdot)$ is a bi-linear operator. An entry $\boldsymbol{A}[i][j]$ is typically non-zero if the global basis-functions Φ_i and Φ_j are coupled, i.e. they have overlapping support (that is if we do not taking into account the possible orthogonality of the basis). The table below summarises the coupling between the different modes:

	a_i (number of modes)	b_i (number of modes coupled to)
interior vertex modes	$(\lvert\mathcal{E}^{1d}\rvert - 1)^2$	$(2N-1)^2$
interior edge modes	$2\lvert\mathcal{E}^{1d}\rvert(\lvert\mathcal{E}^{1d}\rvert - 1)(N-2)$	$N(2N-1)$
interior modes	$\lvert\mathcal{E}^{1d}\rvert^2(N-2)^2$	N^2
boundary vertex modes	$4(\lvert\mathcal{E}^{1d}\rvert - 1)$	$N(2N-1)$
boundary edge modes	$4\lvert\mathcal{E}^{1d}\rvert(N-2)$	N^2
boundary corner modes	4	N^2
$nnz = \sum_i a_i b_i$	$\left[\lvert\mathcal{E}^{1d}\rvert(N^2-1)+1\right]^2$	

As a result, when using a sparse matrix storage format, the global matrix evaluation strategy requires $\left[|\mathcal{E}^{1d}|(N^2-1)+1\right]^2$ floating point multiplications and $\left[|\mathcal{E}^{1d}|(N^2-1)+1\right]^2$ floating point additions leading to a total operation count of $2\left[|\mathcal{E}^{1d}|(N^2-1)+1\right]^2 = 2\left[|\mathcal{E}^{1d}|P(P+2)+1\right]^2$.