An ultraspherical spectral element method for solving partial differential equations

by

Emma Alida Nel

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Applied Mathematics) in the Faculty of Science at Stellenbosch University

Supervisor: Prof. N. Hale

December 2023

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2023

Abstract

We investigate the ultraspherical spectral element method for solving second-order partial differential equations in two dimensions. Moreover, a novel coordinate transformation is introduced to broaden the scope of the method, making it applicable to rectangular domains with circular holes (square donuts), as well as certain types of curved boundaries. The presented method is an integration of two approaches, namely the ultraspherical spectral method and the hierarchical Poincaré–Steklov (HPS) scheme. The ultraspherical method is a Petrov–Galerkin scheme that presents operators in the form of sparse and almost-banded matrices, enabling both stability and computational efficiency. The HPS method is a recursive domain decomposition strategy that enables fast direct solves. It merges solution operators and Dirichlet-to-Neumann operators between subdomains, enforcing continuity of the solution and its derivative across domain boundaries. The fusion of these two methods, combined with a bilinear mapping, results in an accurate discretisation with an explicit direct solve that can be applied to problems on arbitrary polygonal domains with smooth solutions. A major advantage is the reuse of precomputed solution operators facilitated by the HPS scheme, enhancing the efficiency of elliptic solves within implicit and semi-implicit time-steppers. Additionally, the approach is highly parallelisable, allowing for efficient computation time. An implementation of the method is established as a software system, `ultraSEM`, which employs the HPS method to solve on rectangular and polygonal domains. We extend this implementation to allow for solving on domains with circular cavities. This extension relies on a nonlinear coordinate mapping and proves to work effectively, achieving near machine level precision accuracy. On some simple test problems, we demonstrate geometric convergence for refinement of the polynomial degree and algebraic convergence for domain refinement. Furthermore, we show that execution times scale comparably to those achieved for a rectangular domain. We demonstrate the application of the method on various time-dependent and fluid dynamics examples, including contaminant transport and reaction-diffusion systems, and underscore the practical applicability of the methodology and the new domain.

Opsomming

Ons ondersoek die ultrasferiese spektrale element metode vir die oplossing van tweede-orde parsiële differensiaalvergelykings in twee dimensies. 'n Nuwe koördinaattransformasie word voorgestel om die omvang van die metode te verbreed en dit van toepassing te maak op reghoekige gebiede met sirkelvormige gate, sowel as sekere tipes gekurfde grense. Die voorgestelde metode is 'n integrasie van twee benaderings, naamlik die ultrasferiese spektrale metode en die hiërargiese Poincaré–Steklov (HPS) skema. Die ultrasferiese metode is 'n Petrov–Galerkin-skema wat operatore skep in die vorm van matrikse wat effektief is om mee te werk en goeie stabiliteit bied. Die HPS-metode is 'n rekursiewe gebied-ontbindingstrategie wat vinnige direkte oplossings moontlik maak. Dit weef oplossings-operatore en Dirichlet-na-Neumann-operatore tussen subgebiede saam. In die proses word die kontinuïteit van die oplossing en sy afgeleide, oor gebiedsgrense afgedwing. Die samesmelting van hierdie twee strategieë, gekombineer met 'n bilineêre afbeelding, lei tot 'n direkte strategie om akkurate oplossings te genereer vir probleme op veelhoekige gebiede met gladde oplossings. 'n Groot voordeel is die hergebruik van voorafberekende oplossings-operatore wat deur die HPS-skema gefasiliteer word. Dit verbeter veral die doeltreffendheid om elliptiese probleme op te los binne implisiete en semi-implisiete tydstappers. Daarbenewens is dit ook moontlik om die benadering in parallel toe te pas en dus 'n doeltreffende berekeningstyd moontlik te maak. 'n Reedsbestaande implementering van die metode bestaan wel as 'n sagtewarestelsel genaamd `ultraSEM`. Hierdie sagteware gebruik die HPS-metode om oplossings op reghoekige en veelhoekige gebiede te vind. Ons brei hierdie implementering uit om die oplossing op gebiede met sirkelvormige holtes moontlik te maak. Hierdie uitbreiding maak staat op 'n nie-lineêre koördinaat afbeelding en word bewys om effektief te werk. Met 'n paar eenvoudige toetsprobleme word die metode gedemonstreer. Dit vertoon geometriese konvergensie wanneer die polinoomgraad verfyn word en algebraïese konvergensie vir gebiedverfyning. Verder groei die berekeningstyd teen 'n vergelykbare tempo as dit wat vir 'n vierhoekige gebied behaal word. Ons demonstreer die toepassing van die nuwe metode op verskeie tydafhanklike en vloeidinamika voorbeelde, insluitend kontaminant vervoer en reaksie-diffusie stelsels. Met hierdie voorbeelde word die praktiese toepaslikheid van die metodologie en die nuwe gebied onderstreep.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Nick Hale, for his invaluable guidance, mentorship, and expertise throughout the entire research process. Nick's quick wit, clever problem-solving, and unwavering willingness to assist have made the experience of working with him genuinely delightful. His insightful feedback and constructive criticism have also played a pivotal role in shaping the quality of this thesis. I express my appreciation to his dog, Roxy, whose tail wags and cheerful greetings served as emotional support, helping me to persevere throughout this project.

I am indebted to the Wilhelm Frank Trust for their generous support. Their financial assistance enabled me to concentrate on my research and academic pursuits with greater focus and dedication.

Furthermore, I would like to acknowledge my fellow students and lecturers at the Applied Mathematics division of Stellenbosch University. Their presence and encouragement fostered a supportive environment in which I could freely seek guidance, ask questions, and remain motivated to complete the work. Their understanding and acceptance of the inevitable ups and downs of the research process allowed me to progress with confidence and resilience.

Finally, I would like to express my sincere appreciation to my partner, Louis, as well as my friends and family, who demonstrated remarkable patience while I frequently complained about this thesis.

Dedication

In memory of my mom, whose pride in this work would have meant the world to me.

Your absence is felt every day.

CONTENTS

CHAPTER 1

INTRODUCTION

Ordinary and partial differential equations (ODEs and PDEs) are fundamental tools in the modelling of physical systems that evolve in space and/or time. They provide a framework for understanding a wide range of phenomena in fields such as fluid dynamics, oceanography, aerodynamics, electrostatics, thermodynamics, image processing, and topology [1, 8, 10, 69, 75].

When modelling real-life phenomena with ODEs and PDEs, problems arise where analytical solutions are either unavailable or intractable due to the complexity of the equations [9]. In many such cases, one turns to numerical methods to find approximate solutions. Typically, these numerical approaches entail discretising the equation and ultimately solving large systems of linear equations. Methods for solving linear systems of equations fall into one of two categories: direct or iterative methods. While iterative techniques tailored for specific problems can outperform direct methods in certain scenarios, they lack the generality and robustness that direct methods provide [32]. Interest in fast direct solvers has significantly grown in recent years, largely fuelled by the advancements in computational capabilities and fast direct methods over the past two decades [27]. Despite the increased computational resources, the use of direct methods for large-scale computations necessitates efficient algorithms that capitalise on sparse and structured matrices.

In this project, spectral methods are presented as an appropriate method for solving ODEs and PDEs since they are known for providing accurate solutions and stable algorithms [8, 75]. However, two obstacles remain when it comes to developing an efficient, yet general, spectral solver. The first is that many spectral techniques are based on dense matrices, leading to expensive computations [8, 75]. Secondly, spectral methods are typically only applied to regular domains, making it difficult to design a solver that can handle problems on any geometry [60].

In this work, we combine two recent strategies, namely the ultraspherical spectral method [58, 73] and the hierarchical Poincaré–Steklov method [4, 28, 29, 52], to overcome some of these challenges. The ultraspherical method is recognised for producing sparse matrices, ensuring reliability and stability, and offering efficient computational complexity. Partitioning the domain into smaller elements extends the applicability of the method. The hierarchical Poincaré-Steklov scheme is then used as a convenient and efficient direct method to recombine the partitions. To expand this capability even further, we aim to develop a new domain, with a novel mapping, which is built into an existing implementation.

By focusing on the integration of these two strategies, this thesis aspires to bridge gaps and extend the reach of spectral methods to a broader spectrum of practical applications.

1.1 Aims and objectives

This thesis aims to explore an existing direct spectral method and extend the implementation thereof to incorporate a new domain. This new domain is designed to address problems on rectangular geometries with circular holes, such as a perforated plate.

There are several building blocks in achieving this aim. A first objective is to explore, understand, and implement the ultraspherical spectral method for solving ODEs and PDEs on regular domains, as introduced by Olver and Townsend [58, 73]. This includes investigating its application to one- and two-dimensional problems, and approaches for applying it to time-dependent problems, systems of equations, and nonlinear problems.

Building on this foundation, the second objective is to incorporate the use of a domain decomposition technique, known as the hierarchical Poincaré–Steklov method, to transform the solver into a spectral element method. This includes an in-depth investigation into the structure of an existing implementation by Fortunato et al. [23], based on the work of Gillman and Martinsson [28, 29, 52].

With all the structures in place, the final objective is to design and implement a new domain, based on a novel mapping, that extends the applicability of the existing method. This domain will allow the software to solve problems on geometries with circular holes and certain domains with curved boundaries.

1.2 Project outline

This thesis is structured as follows. In Chapter 2, we commence by providing an overview of existing spectral methods for solving ODEs and PDEs. Chapter 3 follows, wherein the essential operators for the global ultraspherical spectral method is constructed in both one and two dimensions. We supplement this with practical examples for better clarification. Additionally, we explore approaches for applying this technique to address time-dependent problems, systems of equations, and nonlinear problems. Subsequently, Chapter 4 explores the necessity of domain decomposition strategies, and introduces a specific approach, called the hierarchical Poincaré–Steklov method. This method allows for solving on irregular geometries by decomposing them into rectangular subdomains, effectively breaking down large problems into more manageable pieces. We investigate transformations essential for accommodating non-rectangular geometries in Chapter 5. Extending that work, we develop a novel mapping to facilitate computation within domains containing curved boundaries, thus enhancing the method's adaptability. We then showcase the versatility of this new transformation through its application to various practical problems in Chapter 6. To validate the method, an in-depth analysis of results obtained from test problems is presented, underlining its effectiveness and practical utility. We conclude in Chapter 7 by summarising our findings and proposing potential directions for future research and advancements in the field.

CHAPTER 2
BACKGROUND

2.1 Existing methods for ODEs and PDEs

When simulating real-world phenomena using ODEs and PDEs, challenges arise due to the complexity of the equations or the domain geometry, making analytical solutions either impractical or unavailable. Typically, numerical methods offer an avenue to approximate solutions in such scenarios. Numerical methods work by discretising a continuous problem as a discrete set of equations that can be solved using computational techniques. Among the numerous classes of numerical methods available for solving two-dimensional ODEs and PDEs, three stand out as the most widely used: finite difference methods, finite element methods, and spectral methods. Each of these approaches possess distinct strengths and weaknesses, and the choice of the best method depends on the specific problem at hand. Additional factors, such as the complexity of the domain, available computational resources, and the required level of accuracy, play significant roles in determining the most suitable approach.

Finite difference methods (FDM) offer good computational efficiency by utilising local approximations and low-order polynomials [9, 47, 49]. In FDM, the domain is typically discretised into a grid of points at which an approximate solution is sought. Derivatives at these grid points are approximated by using finite difference formulas, and the differential equation is transformed into a system of algebraic equations, often involving sparse matrices. Due to their straightforward nature, FDM are relatively easy to implement and find common usage in problems on regular grids with simple geometries. They serve as valuable tools widely employed in computational fluid dynamics, heat transfer, and other fields [48, 55]. Nonetheless, their simplicity often leads to approximate solutions with limited accuracy, and they can pose challenges when applied to complicated geometries [8].

Finite element methods (FEM) are versatile numerical techniques widely employed to solve ODEs and PDEs [18, 20, 41, 47]. In these methods, a *weak form* of the differential equation is constructed by multiplying it with a test function and integrating over elements. The test function allows the differential equation to be satisfied in an average or weighted sense over each element, rather than pointwise. Local solutions are approximated on each of the elements using piecewise polynomials, referred to as shape functions. A global system of algebraic equations is then formed by assembling the weak form equations across elements. The decomposition into multiple elements provides FEM with the flexibility to handle complex geometries and irregularly shaped domains, making it ideal for structural analysis, solid mechanics, and fluid dynamics applications [34, 60]. A typical example of its application is

the modelling of the shell of an auto-mobile. However, FEM's drawback lies in its relatively low accuracy due to the use of low-degree polynomials on the elements [8].

One of the key distinctions between spectral methods and FEM lies in the degree of the polynomials used to approximate solutions. While FEM, similar to FDM, employ *low*-degree polynomials and *local* approximations on each element, spectral methods utilise *global* functions of *high* degree [8]. This distinction is illustrated in Figure 2.1.

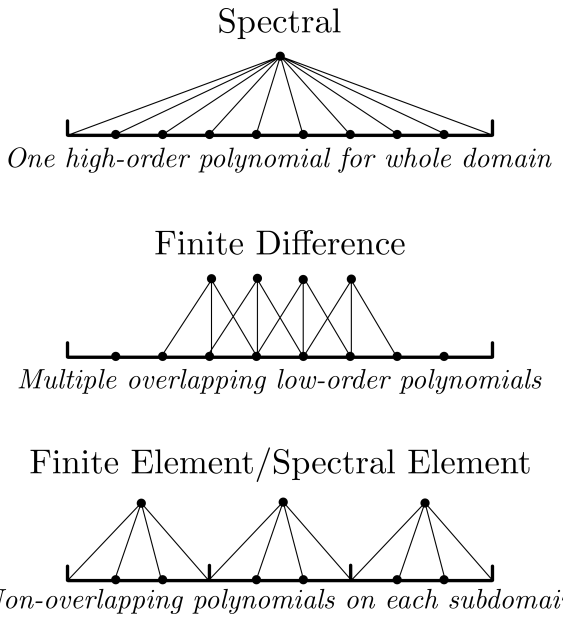

Figure 2.1: This visual representation contrasts three types of numerical algorithms for solving differential equations. Finite difference methods exhibit a localised nature, where the approximation at a specific point is influenced solely by neighbouring point values. On the contrary, spectral methods have a global influence, with the solution at a point being impacted by values across the entire domain. In finite element methods and spectral element methods, approximations in a particular subdomain are influenced only by points within that subdomain. Source: Adapted from [8].

Spectral methods (SM) employ polynomials of high degree across the entire computational domain [10, 45, 62, 75]. The discretisation process involves working with either values at grid points (nodal) or coefficients in a series expansion (modal). In both scenarios, differentiation matrices are employed for approximating derivatives. While it is true that these matrices are typically dense due to the global approach inherent to SM, the pay-off is generally high accuracy. Spectral methods often attain ten digits of accuracy in scenarios where FDM or FEM typically reach only two or three digits [62]. Moreover, at a lower accuracy, spectral methods often demand less computer memory compared to other approaches [75].

Another advantage of SM is that they are defined throughout the entire computational domain, thus enabling easy evaluation of the function under consideration at any point within the domain. This feature proves especially valuable when graphical representations of the solution are needed. Additionally, assessing the accuracy of coefficient based spectral methods becomes straightforward by examining the decrease of the coefficients. This obviates the need for multiple calculations with varying resolutions, as is commonly done in FDM and similar methods to estimate "grid-convergence" [45].

Spectral element methods (SEM) were developed with the aim of combining the domain flexibility provided by FEM with the accuracy characteristics of SM [10, 39, 40]. In SEM, the domain is partitioned into elements, similar to finite elements, while employing sufficiently high-degree polynomials. SEM typically employ the weak form of the differential equation, distinct from the strong form traditionally used in SM. Nevertheless, a significant portion of the underlying theory in SEM closely resembles that of global SM [8]. In the upcoming sections, our focus will primarily revolve around explaining global SM concepts, with a more detailed exploration of domain decomposition into elements to follow in Chapter 4.

When dealing with smooth solutions and simple geometries, spectral methods are usually the preferred choice [8, 62, 75]. In such circumstances, spectral methods are known to offer heightened efficiency and achieve considerably higher accuracy compared to alternative approaches, all the while remaining relatively straightforward to implement. In complex geometries, transitioning to spectral element techniques proves beneficial [39]. For this reason, the spectral technique employed in this project represents a hybrid approach, sharing characteristics with both global SM and SEM. It involves partitioning domains into elements while still operating within the framework of the strong form of the equation.

2.2 An overview of spectral techniques

Spectral methods encompass various classifications, with 'Galerkin', 'tau', and 'collocation' (or 'pseudospectral') methods being the most common distinctions [75]. The former two methods fall into the category of *modal* approximations, since they operate with the coefficients of a global expansion in some prescribed basis. Whereas the latter is *nodal* in nature, dealing with values at specific nodes or points, referred to as collocation points [45].

In collocation methods, the approximate solution is tailored to satisfy the differential equation precisely at appropriately chosen collocation points. To achieve this, a polynomial interpolant is constructed using the unknown solution values at these points and a differentiation matrix is generated to calculate approximate derivative values. Because the collocation points correspond to physical positions in space, it is conceptually straightforward to visualise and address aspects such as boundary conditions [51]. The downside of collocation strategies are that the differentiation matrices are typically dense, leading to similarly dense resulting operators, which are often ill-conditioned [75].

On the other hand, Galerkin-type spectral methods often employ sparse matrices, resulting in faster computations compared to the collocation strategy [8, 75]. In this approach, the polynomial interpolant is approximated by a linear combination of basis functions or modes. These basis functions are usually chosen to be orthogonal, such as trigonometric functions, Legendre polynomials, or Chebyshev polynomials. The solution is then represented by the coefficients in this linear combination [10, 31].

The tau method is a variant of the Galerkin method designed to enhance the accuracy of the solution even further [45]. It introduces a correction term, known as 'tau', into the Galerkin formulation, allowing the use of basis functions that do not satisfy the homogeneous boundary conditions. Tau methods are particularly used for specific nonlinear problems where Galerkin methods are insufficient [8].

Due to their advantage of employing sparse matrices, offering faster computations and better conditioning, the solver in this project is built on Galerkin methods, which will be the focus of the coming sections.

2.3 The spectral Galerkin approach

In the development of a Galerkin method, the choice of basis functions (or modes) is an important consideration. Certain choices of expansion functions offer more advantages than others [16]. When it comes to approximating a function as a series expansion, there are three key properties that the basis functions should possess: (i) being easy to evaluate, (ii) rapid convergence of expansion coefficients, and (iii) representing any solution to arbitrarily high accuracy by taking enough terms in the expansion [8, 45].

In solving one-dimensional (1D) periodic problems, the Fourier basis emerges as an appropriate choice. The Fourier series exhibits rapid convergence, especially for infinitely differentiable functions, and benefits from the efficiency of the fast Fourier Transform (FFT) algorithm for the calculation of sums or coefficients [8]. However, its effectiveness wanes when dealing with nonperiodic problems, as it typically introduces Gibbs oscillations at the domain boundaries [62]. Alternative basis functions are needed for nonperiodic domains.

Both Chebyshev and Legendre polynomials are commonly favoured in Galerkin spectral methods for nonperiodic domains due to their rapid convergence and numerical stability [76]. However, the Chebyshev polynomials can be viewed as a cosine Fourier series in disguise. Consequently, it shares some valuable properties with the Fourier basis, such as the utilisation of the FFT algorithm, and more specifically the Discrete Cosine Transform. This feature allows for optimised and efficient evaluations. Algorithms are available that make fast computations practicable for Legendre interpolants as well [64]; however, Chebyshev remains the more straightforward case. There are other types of spectral Galerkin methods, such as radial basis functions [7] and wavelet-based methods [56], but we do not consider those here.

2.3.1 Chebyshev polynomials and points

The Chebyshev polynomials *of the first kind* $T_k(x)$ are the polynomials of degree k defined by

$$T_k(x) = \cos(k \cos^{-1}(x)), \qquad k = 0, 1, 2, \ldots, \tag{2.1}$$

and which are orthogonal on $[-1, 1]$ with respect to the weight function $w(x) = (1-x^2)^{-1/2}$ [57]. Figure 2.2 displays the first few polynomials, which satisfy the recurrence relation,

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{k+1}(x) &= 2xT_k(x) - T_{k-1}(x), \quad k = 1, 2, \ldots \end{aligned} \tag{2.2}$$

The *Chebyshev points of the second kind*, alternatively known as Gauss–Lobatto, Chebyshev extrema, or simply Chebyshev points, are given by

$$x_j = \cos\left(\tfrac{j\pi}{n}\right), \qquad j = 0, 1, \ldots, n, \tag{2.3}$$

on the interval $[-1, 1]$. These nodes commonly serve as grid points for spectral collocation schemes. Their clustering around the endpoints of the interval, visualised on the right in Figure 2.2, ensures that their polynomial interpolations remain stable, a property that proves beneficial when it comes to approximating solutions [76]. Furthermore, their inclusion of the interval endpoints simplifies the implementation of boundary conditions when solving problems, as we shall see in Section 3.2.1.

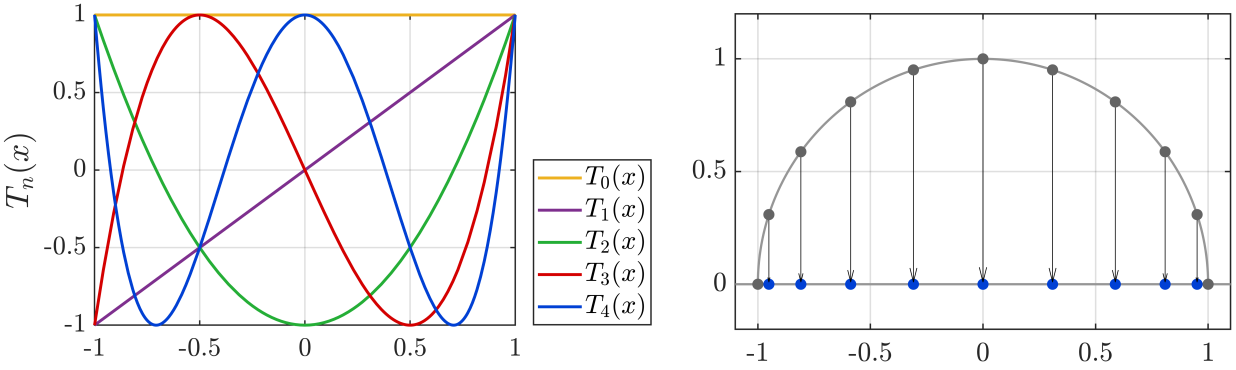

Figure 2.2: The first five Chebyshev polynomials of the first kind are displayed on the left side. On the right, the Chebyshev points of the second kind are shown for $n = 10$, calculated using formula (2.3).

2.3.2 Approximating functions

The basis of the spectral methods we consider in this thesis is the approximation of solutions as linear combinations of Chebyshev polynomials. Theorem 1 is helpful in this regard.

Theorem 1. *[76, Thm 3.1] If f is Lipschitz continuous on $[-1, 1]$, it has a unique representation as a Chebyshev series,*

$$f(x) = \sum_{k=0}^{\infty} a_k T_k(x),$$

which is absolutely and uniformly convergent, and the coefficients are given for $k \geq 1$ by the formula

$$a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx,$$

and for $k = 0$ by the same formula with the factor $2/\pi$ changed to $1/\pi$.[1]

[1] A function f is Lipschitz continuous on a domain Ω if there is a constant C such that $|f(x) - f(y)| \leq C\,|x - y|$ for all $x, y \in \Omega$.

We consider two approaches to approximate such a function f: truncation and interpolation [76]. The approximation to f obtained by truncation or projection of the series to degree n is given by the polynomial

$$f_n(x) = \sum_{k=0}^{n} a_k T_k(x),$$

where the first $n+1$ coefficients are the same as those of f itself. The alternative, which is to approximate f by the degree n polynomial obtained by interpolation in the Chebyshev points, is

$$p_n(x) = \sum_{k=0}^{n} c_k T_k(x), \quad \text{such that} \quad p_n(x_j) = f(x_j), \quad j = 0, 1, \ldots, n+1.$$

Both the interpolation strategy with the polynomials $\{p_n\}$, and the truncation approach with the polynomials $\{f_n\}$, provide good approximations to f. This is reinforced by Theorem 2, which asserts that if f is analytic on $[-1, 1]$, then both $\|f - f_n\|_\infty$ and $\|f - p_n\|_\infty$ decrease *geometrically* to 0 as $n \to \infty$.[2] Geometric convergence means that the error decreases at the rate $\mathcal{O}(C^{-n})$ for some constant $C > 1$.

The concept of a Bernstein ellipse is needed for Theorem 2. A Bernstein ellipse, denoted by E_ρ, is an ellipse in the complex plane with foci at -1 and 1, with ρ equal to the sum of the lengths of the semimajor and semiminor axes. Typically, $\rho > 1$.

Theorem 2. *[76, Thm 8.2] Let a function f analytic in a neighbourhood of $[-1, 1]$ be analytically continuable to an open Bernstein ellipse E_ρ, where it satisfies $|f(z)| \leq M$ for some M. Then for each $n \geq 0$ its Chebyshev truncations f_n satisfy*

$$\|f - f_n\|_\infty \leq \frac{2M\rho^{-n}}{\rho - 1}.$$

and its Chebyshev interpolants p_n satisfy

$$\|f - p_n\|_\infty \leq \frac{4M\rho^{-n}}{\rho - 1}.$$

Theorem 2 further supports the Chebyshev polynomials as an appropriate choice for the expansion functions. The fast convergence of the Chebyshev basis for analytic functions is not only demonstrated by the rapidly converging error according to Theorem 2, but also by the rapid decay of the Chebyshev coefficients in an expansion. Theorem 3 gives a theoretical upper bound on the value of the k-th Chebyshev coefficient of an analytic function. The observed geometric convergence demonstrates that a solution can be accurately represented by a Chebyshev series when using a sufficient number of terms in the expansion.

[2] The notation $\|f\|_\infty$ represents the infinity norm or supremum norm of the continuous function f, calculated as the maximum absolute value on a closed and bounded interval.

Theorem 3. *[76, Thm 8.1] If f has the properties of Theorem 2, then its Chebyshev coefficients satisfy*

$$|a_k| \leq 2M\rho^{-k},$$

with $|a_0| \leq M$ in the case $k = 0$.

As an example, consider the Runge function $f(x) = (1 + 25x^2)^{-1}$, which has poles at $x = \pm\frac{i}{5}$ and is thus not analytic at those points. The largest Bernstein ellipse in which the function is analytic has a semiminor axis of 0.2 and a semimajor axis of approximately 1.02, and is sketched on the left in Figure 2.3. When the Runge function is approximated by an interpolant expanded as a Chebyshev series, the expected geometric convergence from Theorem 2 is observed. We also see the rapid decrease in the magnitude of the coefficients in the expansion predicted by Theorem 3. This is demonstrated in Figure 2.3.

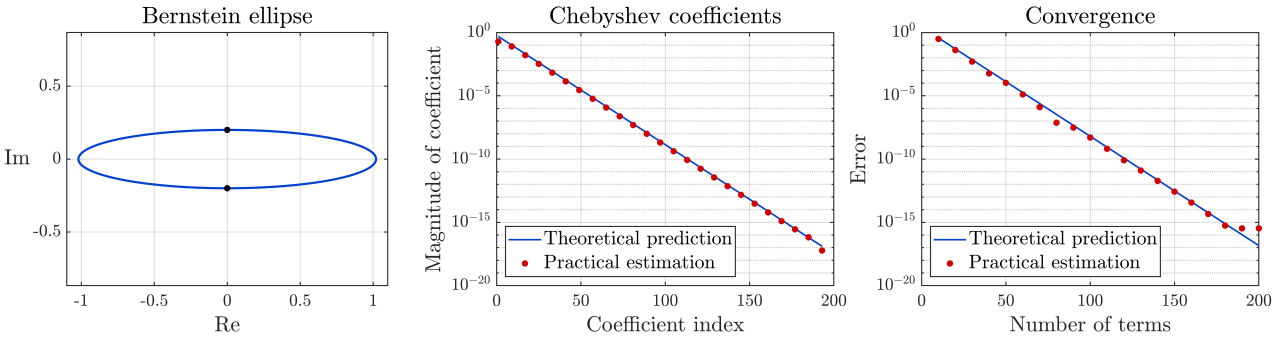

Figure 2.3: The image on the left displays the Bernstein ellipse E_ρ with a value of $\rho \approx 1.22$. The ellipse has foci at -1 and 1, and its semiminor axis measures 0.2. This is the largest ellipse contained within the analyticity of the Runge function in the complex plane, thus governing its convergence, as outlined in Theorems 2 and 3. In the centre image, we observe the magnitude of Chebyshev expansion coefficients a_k, satisfying the theoretical upper bound specified by Theorem 3. On the right, the practical error is presented, calculated as the infinity norm of the difference between the Chebyshev expansion and the Runge function. A comparison with the theoretical convergence rate predicted by Theorem 2 reveals a matching slope. The presence of a plateau is attributed to rounding errors at around the level of machine precision.

There exists ample evidence supporting the effectiveness of the Chebyshev basis in approximating analytic functions. However, it is worth considering the expected performance when dealing with non-analytic functions. The Jackson theorems [13] established a bound for polynomial approximations of non-analytic functions. It asserts that if a function f is ν times continuously differentiable on $[-1, 1]$, then its best polynomial approximations converge at the algebraic rate $\mathcal{O}(n^{-\nu})$.

However, this bound can be better addressed for Chebyshev polynomials by using a concept called bounded variation. A function is deemed to have bounded variation if its total variation, V, is finite [76].[3] The finiteness of V allows Chebyshev approximations to converge at a rate faster than predicted by the Jackson theorems. An improved bound is given by Theorem 4.

[3]The total variation of a function f is the 1-norm of the derivative, $V = \|f'\|_1$.

Theorem 4. *[76, Thm 7.2] For an integer $\nu \geq 1$, let f and its derivatives through $f^{(\nu-1)}$ be absolutely continuous on $[-1, 1]$ and suppose the νth derivative $f^{(\nu)}$ is of bounded variation V. Then for any $n > \nu$, its Chebyshev projections, f_n, and its Chebyshev interpolants, p_n, satisfy*

$$\|f - f_n\|_\infty \leq \frac{2V}{\pi\nu(n-\nu)^\nu}, \quad \text{and} \quad \|f - p_n\|_\infty \leq \frac{4V}{\pi\nu(n-\nu)^\nu}.$$

It is evident that smoother functions exhibit faster convergence of their approximations. To demonstrate this in the context of a non-analytic function, we consider $f(x) = |\sin(5x)|^3$. The third derivative of f has discontinuities at $x = 0$ and $x = \pm\pi/5$, implying that it is three times differentiable, but only two times *continuously* differentiable. As per the Jackson theorems, this would predict convergence of order two. The convergence of the Chebyshev interpolant p_n of the function f is depicted on the right in Figure 2.4. On the same graph, the theoretical bound for p_n is shown, which is computed using a total variation of $V \approx 16500$ for the third derivative of f. It is clear that in this example even the stricter bound provided by Theorem 4 is a conservative estimation of the convergence, which is at least of order three. On the left in Figure 2.4, the function f is plotted together with its 50 term Chebyshev approximation.

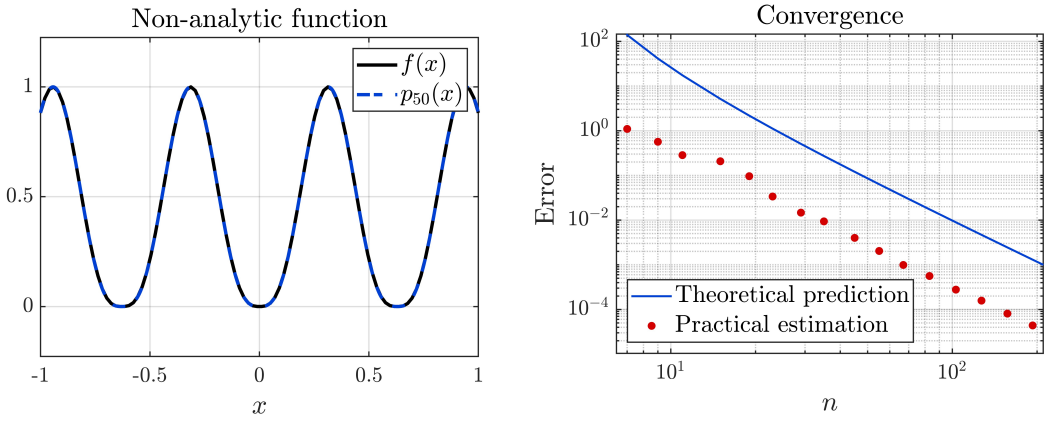

Figure 2.4: The image on the left displays the non-analytic function $f(x) = |\sin(5x)|^3$, together with its 50 term Chebyshev approximation, $p_{50}(x)$. On the right, the infinity norm of the error of this approximation is depicted. This practical error demonstrates a cubic convergence rate, which aligns with the slope of the theoretical bound provided by Theorem 4 and shown by a blue line. This convergence is faster than the quadratic convergence suggested by the Jackson theorems.

2.3.3 Calculation of Chebyshev coefficients

Consider the degree n polynomial interpolant to a function $f(x)$

$$f(x) \approx p_n(x) = \sum_{k=0}^{n} c_k T_k(x).$$

When evaluating $f(x)$ at a single point, x_0, the approximation can then be expressed as:

$$f(x_0) = c_0 T_0(x_0) + c_1 T_1(x_0) + c_2 T_2(x_0) + \ldots + c_n T_n(x_0).$$

Extending this to a set of distinct points $x_0, x_1, \ldots, x_n$, yields the square matrix equation:

$$\begin{bmatrix} T_0(x_0) & T_1(x_0) & \ldots & T_n(x_0) \\ T_0(x_1) & & & T_n(x_1) \\ \vdots & & & \vdots \\ T_0(x_n) & T_1(x_n) & \ldots & T_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

The matrix on the left is known as the *Chebyshev–Vandermonde matrix*. Determining the coefficients c_k from the function values $f(x_k)$ requires the inverse of this matrix, which typically has complexity $\mathcal{O}(n^3)$.[4] On the other hand, determining the function values when the coefficients are known involves multiplication by this matrix, which has a computational complexity of $\mathcal{O}(n^2)$.

However, if the points $\{x_j\}_{j=0}^n$ are strategically chosen, for example, the Chebyshev points as given in formula (2.3), then eq. (2.1) can be utilised to simplify the entries in the matrix to:

$$T_k(x_j) = \cos\Big(k \cos^{-1}\big(\cos\big(\tfrac{j\pi}{n}\big)\big)\Big) = \cos\big(\tfrac{kj\pi}{n}\big).$$

With the expansion functions now simplified to cosine functions, the conversion between function evaluations and expansion coefficients can be performed via the Discrete Cosine Transform (DCT). This transformation improves the computational complexity of switching between coefficients and values to $\mathcal{O}(n \log n)$ [61].

2.3.4 Ultraspherical polynomials

When employing spectral methods to solve differential equations, a crucial consideration is the computation of derivatives. The derivative of a Chebyshev polynomial can be expressed in terms of the Chebyshev polynomials themselves [53],

$$\frac{dT_k(x)}{dx} = 2k \sum_{\substack{r=0 \\ k-r \text{ odd}}}^{k-1} T_r(x),$$

where the factor $2k$ is changed to k when $r = 0$. However, implementing this formula results in an upper triangular differentiation matrix.[5] For a more efficient approach, an alternative is to express the derivative in terms of the *ultraspherical polynomials* [8, 75], which will produce a banded differentiation matrix.

The ultraspherical polynomials or *Gegenbauer polynomials*, denoted as $C_k^{(\lambda)}(x)$, are a set of orthogonal polynomials defined on the interval $[-1, 1]$ with respect to the weight function $w(x) = (1 - x^2)^{\lambda - 1/2}$ [57].

[4] In practice, instead of inverting the matrix directly, one would use an operator in standard software for solving linear systems, such as 'backslash' in MATLAB.

[5] Further elaboration on the construction of the differentiation matrix will be provided in Section 3.1.1.

These polynomials are characterised by the recurrence relation:

$$\begin{aligned} C_0^{(\lambda)}(x) &= 1 \\ C_1^{(\lambda)}(x) &= 2\lambda x \\ C_k^{(\lambda)}(x) &= \frac{1}{k}\Big[2x(k+\lambda-1)C_{k-1}^{(\lambda)}(x) - (k+2\lambda-2)C_{k-2}^{(\lambda)}(x)\Big], \end{aligned} \tag{2.4}$$

with $k \geq 2$. Here, $\lambda \geq 1$ is a real parameter, known as the order of the ultraspherical polynomial. Notably, when $\lambda = 1$, these polynomials become the Chebyshev polynomials *of the second kind*.

The derivatives of the Chebyshev polynomials can be elegantly expressed in terms of the ultraspherical polynomials of the first order:[6]

$$\frac{dT_k}{dx} = \begin{cases} kC_{k-1}^{(1)}(x), & k \geq 1 \\ 0, & k = 0. \end{cases} \tag{2.5}$$

This relationship proves useful in deriving a simple expression for the derivative of a Chebyshev series, which will be an improvement on classical spectral methods which represented differentiation as dense operators [8, 75]. Additionally, an almost equally simple recurrence relation exist that describes the Chebyshev polynomials in terms of the ultraspherical polynomials of the first order [57],

$$T_k(x) = \begin{cases} \frac{1}{2}\left(C_k^{(1)}(x) - C_{k-2}^{(1)}(x)\right), & k \geq 2, \\ \frac{1}{2}C_1^{(1)}, & k = 1, \\ C_0^{(1)}, & k = 0, \end{cases} \tag{2.6}$$

which facilitates easy conversion between the two bases. If higher order derivatives are involved in an equation, then the formula for the λth derivative of the Chebyshev polynomials proves to be helpful [57],

$$\frac{d^\lambda T_k}{dx^\lambda} = \begin{cases} 0, & 0 \leq k < \lambda, \\ 2^{\lambda-1}k(\lambda-1)!\, C_{k-\lambda}^{(\lambda)}, & k \geq \lambda, \end{cases} \tag{2.7}$$

where $C_k^{(\lambda)}$ is the degree-k ultraspherical polynomial of order $\lambda \geq 1$. The ultraspherical polynomials abide by an analogue of (2.6) for conversion between higher orders:

$$C_k^{(\lambda)}(x) = \begin{cases} \frac{\lambda}{\lambda+k}\left(C_k^{(\lambda+1)} - C_{k-2}^{(\lambda+1)}\right), & k \geq 2, \\ \frac{\lambda}{\lambda+1}C_1^{(\lambda+1)}, & k = 1, \\ C_0^{(\lambda+1)}, & k = 0. \end{cases} \tag{2.8}$$

These simple relationships offer a means to construct sparse operators for the spectral method under consideration in this project. Further elaboration on the construction and application of these operators is presented in Section 3.1.

[6] Hereafter the term 'Chebyshev polynomials' will exclusively denote the polynomials of the first kind.

Representing the solution in one basis while expressing the differential equation in another basis is a defining feature of Petrov–Galerkin methods. Typically, the selection of bases in these methods are influenced by the imposed boundary conditions [33, 68], but this can pose challenges for devising a universal solver. In this project, we prioritise preserving sparsity, aligning with similar methodologies in existing studies [19, 70]. As a result, the bases are contingent on the order of the differential equation, rather than being dictated by the specific boundary conditions [58].

2.3.5 Scaling to other domains

Up to this point, we have discussed only the Chebyshev polynomials and points as they are defined within the range of $[-1, 1]$; nevertheless, they can be employed on intervals other than $[-1, 1]$. To adapt the Chebyshev points to a general finite interval $[a, b]$, an affine change of variables is applied to (2.3), giving [75]:

$$x_j = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{j\pi}{n}\right).$$

When referring to the Chebyshev polynomials on intervals other than $[-1, 1]$, we generally mean adapting or extending their properties to a different interval, rather than the polynomials themselves existing on other intervals in their original form. This involves a transformation to shift and scale the polynomials to fit the desired interval, which is described by

$$\widehat{T}_k(x) = T_k\left(\frac{2x-(a+b)}{b-a}\right).$$

While the original Chebyshev polynomials are specifically defined for $[-1, 1]$, the properties and mathematical principles associated with them can be extended and adapted to other intervals through appropriate transformations. For the remainder of this thesis, $\widehat{T}_k(x)$ indicates a Chebyshev polynomial scaled to the appropriate interval, where the interval itself should be clear from the context.

Given that the ultraspherical polynomials are a generalisation of the Chebyshev polynomials, the same transformation is required to adjust them to a specified interval, and their general properties still hold true.

2.3.6 Tensor product formulation in higher dimensions

Univariate polynomials are commonly used in higher dimensions to approximate solutions due to their adaptability and ease of computation [8]. These univariate polynomials are typically employed in a tensor product fashion, where multidimensional basis functions are constructed as products of univariate polynomials in each dimension. For example, if in two dimensions one has univariate polynomials $\phi_i(x)$ and $\phi_j(y)$, then the bivariate basis functions would be given by $\phi_{ij}(x, y) = \phi_i(x) \cdot \phi_j(y)$.

Note that the univariate polynomials employed in different dimensions do not necessarily have to be of the same type. The selection of polynomial families for each dimension depends significantly on the specific geometry of the problem at hand. For instance, Fourier–Chebyshev bases have been used for solving problems in circular domains using cylindrical coordinates [33]. In such cases, univariate functions in the Fourier basis are used to discretise the periodic component (e.g., θ), while the Chebyshev basis is employed for the nonperiodic component (e.g., radial direction). It remains a challenge to make an optimal choice of basis functions when dealing with irregular domains, however, there have been some recent developments in employing shifted Jacobi polynomials with different parameters to effectively handle triangular domains [59].

In cases where the domain is regular, the Chebyshev polynomials continue to be a favoured choice due to their advantageous features, which includes numerical efficiency, good convergence properties, stability, and the ease of transforming spatial coordinates into coefficients [8]. In the two-dimensional (2D) context, this involves expressing the bivariate basis as a tensor product of Chebyshev series in the x- and y-directions. A function $f(x, y)$ is then approximated as a linear combination of these basis functions:

$$f(x, y) \approx \sum_{i=0}^{n_y} \sum_{j=0}^{n_x} X_{ij} T_i(y) T_j(x).$$

Here, n_x and n_y represent the degrees of the polynomials in each dimension, while X_{ij} constitutes the matrix of coefficients characterising the solution. As in 1D, the DCT can be used to transform between function evaluations and coefficients. This 2D transform requires the sequential application of the 1D transform row-by-row and then column-by-column, leading to a complexity of $\mathcal{O}\big(n^2 \log n\big)$ when $n_x = n_y = n$ [61].

In higher dimensions, the computational cost can increase significantly due to the increased number of basis functions and coefficients. Therefore, efficient algorithms and techniques, such as sparse grids and hierarchical representations, become crucial to manage the complexity and enhance the performance. In the upcoming chapter, we explore the fundamentals of a particular Galerkin method known as the ultraspherical spectral method. This method effectively utilises sparse operators and incorporates specialised techniques to preserve and leverage this sparsity, ultimately enhancing computational efficiency.

CHAPTER 3

THE ULTRASPHERICAL SPECTRAL METHOD

The ultraspherical spectral method was introduced by Olver and Townsend [58]. It is a Petrov–Galerkin scheme that optimally leverages ultraspherical polynomial bases to represent differentiation operators with maximum sparsity. This method capitalises on the insight that the derivatives of Chebyshev polynomials become diagonal when expressed in terms of higher-order ultraspherical polynomials, as described in eq. (2.7). In 1D, the ultraspherical spectral method results in almost-banded linear systems that can be efficiently solved with linear complexity.

Following the initial application of this method to solve linear ODEs, Townsend and Olver extended its utility to linear PDEs defined on rectangular domains [73]. Subsequently, researchers have built upon their work, broadening the method's applicability. These extensions include adapting the method to solve nonlinear time-dependent PDEs [14], refining it for self-adjoint problems [3], and exploring its implementation in spectral element methods [23].

This chapter provides an overview of the fundamental components underlying the ultraspherical spectral method. The tools required for solving linear ODEs are outlined in Section 3.1, with an illustrative example showcased in Section 3.2. In this example, the optimisation of solutions via preconditioners and specialised techniques designed for sparse structures are considered. Additionally, the extension of the method to address time-dependent problems is considered in Section 3.3, as well as applications to systems of differential equations in Section 3.4, and nonlinear problems in Section 3.5. Finally, leveraging tools from preceding sections enables the 2D problem-solving approach for PDEs to be explored in Section 3.6.

3.1 Constructing the operators

Consider the one-dimensional linear ODE of Kth order with variable coefficients defined on $[-1, 1]$ of the form:

$$\sum_{\lambda=0}^{K} a_\lambda(x) \frac{d^\lambda u}{dx^\lambda} = f(x), \tag{3.1}$$

along with K linear constraints. When utilising the ultraspherical spectral method to solve equations such as (3.1), the solution is approximated as a Chebyshev expansion, with the

primary objective being to determine the expansion coefficients u_k:

$$u(x) = \sum_{k=0}^{\infty} u_k T_k(x). \tag{3.2}$$

To this end, we formulate the differentiation of the solution and the multiplication by a variable coefficient in terms of operators acting on these coefficients. The key essence of the ultraspherical spectral method lies in its ability to render these operators sparse, and as a result reduce (3.1) to a sparse system of linear algebraic equations to be solved. Note that the operators developed in the upcoming sections are designed for the interval $[-1, 1]$, as this is the primary focus in our approach. However, they can be applied to different intervals when an appropriate scaling factor is incorporated.

3.1.1 The differentiation operator

The relationship (2.5) from Section 2.3.4 provides a simple expression for the derivative of a Chebyshev polynomial. Implementing this derivative in the differentiation of eq. (3.2) scales the coefficients and changes the basis function as follows:

$$u'(x) = \sum_{k=1}^{\infty} u_k T_k'(x) = \sum_{k=1}^{\infty} u_k k C_{k-1}^{(1)}(x),$$

where $C^{(1)}$ are the ultraspherical polynomials of order $\lambda = 1$. Thus, the vector of coefficients describing the derivative $u'(x)$ in a $C^{(1)}$ series is given by $D_1\boldsymbol{u}$, where D_1 is the banded linear operator,

$$D_1 = \begin{bmatrix} 0 & 1 & & & \\ & 0 & 2 & & \\ & & 0 & 3 & \\ & & & \ddots & \ddots \end{bmatrix},$$

and $\boldsymbol{u}$ is the vector of Chebyshev expansion coefficients of $u(x)$. In the process of discretising higher order derivatives, formula (2.7) for the λth derivative of the Chebyshev polynomials proves to be helpful. Based on this relationship, one can deduce that the λth order differentiation operator takes the form:

$$D_\lambda = 2^{\lambda-1}(\lambda-1)! \begin{bmatrix} \overbrace{0 & \dots & 0}^{\lambda \text{ times}} & \lambda & & & \\ & & & \lambda+1 & & \\ & & & & \lambda+2 & \\ & & & & & \ddots \end{bmatrix}, \tag{3.3}$$

for $\lambda \geq 1$. For the case of $\lambda = 0$, we set D_0 to be the identity matrix. The matrix D_λ transforms a Chebyshev coefficient vector into a vector containing $C^{(\lambda)}$ coefficients corresponding to the λth derivative. Note that this differentiation operator is sparse and banded, making it an improvement over the conventional dense Chebyshev differentiation matrix commonly employed in spectral collocation methods [75].

3.1.2 The multiplication operator

In addressing the variable coefficients within eq. (3.1), it is helpful to find a representation that treats the multiplication of two Chebyshev series as an operator acting upon coefficients. For this purpose, we initially explore the handling of the most basic, non-constant form of the function $a(x)$, namely $a(x) = x$. By expressing $u(x)$ in terms of its Chebyshev series and then multiplying it with x, one obtains

$$xu(x) = \sum_{k=0}^{\infty} u_k x T_k(x). \tag{3.4}$$

Recalling that $xT_k(x)$ is a component in the recurrence relation depicted in eq. (2.2), one recognises its reordering as:

$$xT_k(x) = \frac{1}{2}\left(T_{k+1}(x) + T_{k-1}(x)\right),$$

which, when substituted into (3.4), yields

$$\begin{aligned} xu(x) &= \sum_{k=0}^{\infty} u_k x T_k \\ &= u_0 x T_0 + \sum_{k=1}^{\infty} u_k \frac{1}{2}(T_{k+1} + T_{k-1}) \\ &= u_0 T_1 + \frac{1}{2}\sum_{k=2}^{\infty} u_{k-1} T_k + \frac{1}{2}\sum_{k=0}^{\infty} u_{k+1} T_k \\ &= u_0 T_1 + \frac{1}{2} u_1 T_0 + \frac{1}{2} u_2 T_1 + \frac{1}{2}\sum_{k=2}^{\infty}(u_{k-1} + u_{k+1}) T_k \\ &= \frac{1}{2} u_1 T_0 + \frac{1}{2}(2u_0 + u_2) T_1 + \frac{1}{2}\sum_{k=2}^{\infty}(u_{k-1} + u_{k+1}) T_k. \end{aligned} \tag{3.5}$$

Consequently, one can construct an operator, denoted by J_0, which maps the Chebyshev coefficients of the function $u(x)$ to the Chebyshev coefficients of $xu(x)$, as defined by

$$J_0 = \frac{1}{2}\begin{bmatrix} 0 & 1 & & & \\ 2 & 0 & 1 & & \\ & 1 & 0 & 1 & \\ & & \ddots & \ddots & \ddots \end{bmatrix}.$$

If $\boldsymbol{u}$ is the Chebyshev coefficients of $u(x)$, then $J_0\boldsymbol{u}$ returns the Chebyshev coefficients of $xu(x)$. This matrix J_0 is commonly referred to as the *Jacobi matrix* and acts as an operator for multiplying by x. Since x^2 can be expressed as the product of x with itself, x^2 can be discretised using J_0^2. The same principle applies to higher powers of x.

If $a(x)$ is a polynomial of degree m, then, by definition, $a(x)$ can be expanded as a power series, i.e., $a(x) = a_0 + a_1 x + a_2 x^2 + \ldots a_m x^m$. Thus, multiplication by $a(x)$ can be discretised

by replacing the powers of x with powers of the Jacobi matrix. However, the evaluation of power series expansions are generally considered to be less robust [8]. Hence, we prefer to express $a(x)$ as a polynomial expansion by utilising, for instance, the Chebyshev polynomials,

$$a(x) = \sum_{k=0}^{m} a_k T_k(x). \tag{3.6}$$

If $a(x)$ is not a polynomial, but rather a general function, then it can be approximated by the degree m polynomial in (3.6), for sufficiently large m. Theorem 3 can assist in estimating m for functions that are analytic or many times differentiable.

To create the multiplication operator for $a(x)$ in this form, each polynomial $T_k(x)$ must be expressed using powers of the Jacobi matrix. These polynomials should then be scaled by their corresponding coefficients and summed, producing the multiplication operator:

$$M_0[a(x)] = \sum_{k=0}^{m} a_k T_k(J_0). \tag{3.7}$$

Thus, $M_0[a]\boldsymbol{u}$ returns the Chebyshev coefficients of $a(x)u(x)$. To calculate this summation efficiently, one can employ *Clenshaw's Algorithm* [15], a recursive technique used to compute the weighted sum of a finite series of functions or matrix functions. It constitutes a broader version of Horner's method and is particularly effective for evaluating linear combinations governed by a three-term recurrence relation.

Algorithm 1 Clenshaw's algorithm for Chebyshev polynomials

$S_{m+1} \leftarrow 0$
$S_m \leftarrow 0$
for k in $m-1, m-2, \ldots, 1$ **do**
  $S_k = a_j I + 2J_0 S_{k+1} - S_{k+2}$
  $S_{k+2} = S_{k+1}$
  $S_{k+1} = S_k$
end for
$M = a_0 I + J S_1 - S_2$

Algorithm 1 outlines the specifics of Clenshaw's summation, particularly tailored to the recurrence relation of Chebyshev polynomials and adapted for use with the Jacobi matrix for x multiplications. The advantage of this approach is that it eliminates direct dependence on the basis functions, obviating the need to compute these functions via their recurrence relations prior to summation. As a result, enhanced computational efficiency is achieved.

The multiplication operator in (3.7) exclusively acts on and returns coefficients in the Chebyshev basis. To address multiplications involving coefficients in a $C^{(\lambda)}$ series, one must formulate a λth order multiplication operator. This would enable one to determine the $C^{(\lambda)}$ coefficients of $a(x)u(x)$ when $u(x)$ is described in the $C^{(\lambda)}$ basis.

Recall the recurrence relation for ultraspherical polynomials given in eq. (2.4), which, for $k \geq 1$, can be reordered as:

$$xC_k^{(\lambda)} = \frac{1}{2}\left(\frac{k+1}{k+\lambda}C_{k+1}^{(\lambda)} + \frac{k+2\lambda-1}{k+\lambda}C_{k-1}^{(\lambda)}\right).$$

This subsequently paves the way to follow the same steps as in eq. (3.5) to compute the λth order Jacobi matrix,

$$J_\lambda = \frac{1}{2}\begin{bmatrix} 2 & \frac{2\lambda}{1+\lambda} & & & \\ 0 & 0 & \frac{1+2\lambda}{2+\lambda} & & \\ & \frac{2}{1+\lambda} & 0 & \frac{2+2\lambda}{3+\lambda} & \\ & & \frac{3}{2+\lambda} & 0 & \ddots \\ & & & \ddots & \ddots \end{bmatrix}.$$

The multiplication operator of order λ can then be computed using Algorithm 1, with J_0 substituted for J_λ, resulting in the general expression:

$$M_\lambda[a(x)] = \sum_{k=0}^{m} a_k T_k(J_\lambda). \tag{3.8}$$

This operator can accept either the function $a(x)$ or its Chebyshev coefficients $\boldsymbol{a} = [a_0, a_1, \ldots a_m]^\top$ as input and will have a bandwidth of m, which is determined solely by the number of expansion terms necessary to accurately approximate $a(x)$. A smoother function $a(x)$ will result in a smaller bandwidth m.

One final ingredient is required. While the differentiation operator alters the basis of the coefficients it operates on, the multiplication operator does not. This gives rise to a complication in differential equations such as $u'' = a(x)u$, as the differential operator on the left transitions the coefficients into a $C^{(2)}$ basis, while the coefficients on the right remain rooted in the Chebyshev basis. To address this discrepancy, the solution necessitates an operator that can map coefficients from the Chebyshev basis to the $C^{(\lambda)}$ basis. These specific operators are known as conversion operators.

3.1.3 The conversion operator

To construct an operator facilitating the conversion between the Chebyshev and ultraspherical bases, one can leverage the recurrence relation (2.6) expressed in Section 2.3.4. With this relation in hand, a few steps can be applied to transform the Chebyshev series expansion of $u(x)$ into an ultraspherical expansion:

$$\begin{aligned} u(x) = \sum_{k=0}^{\infty} u_k T_k(x) &= u_0 T_0(x) + u_1 T_1(x) + \frac{1}{2}\sum_{k=2}^{\infty} u_k \left(C_k^{(1)}(x) - C_{k-2}^{(1)}(x)\right) \\ &= \frac{1}{2}(2u_0 - u_2)C_0^{(1)}(x) + \frac{1}{2}\sum_{k=1}^{\infty}(u_k - u_{k+2})\,C_k^{(1)}(x). \end{aligned} \tag{3.9}$$

This conversion can be encoded in the form of a sparse and banded operator, S_0, that translates a vector of coefficients of a Chebyshev series into coefficients of a $C^{(1)}$ series:

$$S_0 = \frac{1}{2}\begin{bmatrix} 2 & 0 & -1 & & \\ & 1 & 0 & -1 & \\ & & 1 & 0 & \ddots \\ & & & 1 & \ddots \\ & & & & \ddots \end{bmatrix}. \tag{3.10}$$

If higher-order derivatives are involved, it is advantageous to formulate a more comprehensive conversion matrix, denoted as S_λ, which facilitates the transformation of a vector of $C^{(\lambda)}$ coefficients into a vector of $C^{(\lambda+1)}$ coefficients. Utilising the formula (2.8) the same steps as used to construct S_0 can be followed, resulting in the sparse matrix,

$$S_\lambda = \begin{bmatrix} 1 & 0 & -\frac{\lambda}{\lambda+2} & & \\ & \frac{\lambda}{\lambda+1} & 0 & -\frac{\lambda}{\lambda+3} & \\ & & \frac{\lambda}{\lambda+2} & 0 & \ddots \\ & & & \frac{\lambda}{\lambda+3} & \ddots \\ & & & & \ddots \end{bmatrix}. \tag{3.11}$$

This signifies that if $\boldsymbol{u}$ is the Chebyshev expansion coefficients of $u(x)$, then the sequence of matrices $S_{\lambda-1}\dots S_1 S_0 M_0[a(x)]\boldsymbol{u}$ returns the $C^{(\lambda)}$ expansion coefficients of $a(x)u(x)$.

With the differentiation, multiplication, and conversion operators in our toolbox, we can discretise the general ODE provided in (3.1) as:

$$\left(M_K[a_K]D_K + \sum_{\lambda=0}^{K-1} S_{K-1}\dots S_\lambda M_\lambda[a_\lambda]D_\lambda\right)\boldsymbol{u} = S_{K-1}\dots S_0\boldsymbol{f},$$

where $\boldsymbol{f}$ and $\boldsymbol{u}$ contain the Chebyshev coefficients of the function $f(x)$ and the solution $u(x)$, respectively. Since all the operators are banded, the resulting matrix on the left of this equation will be similarly banded. In most instances, the bandwidth is determined by the multiplication operators. If m_λ denotes the number of expansion terms needed to accurately approximate the variable coefficient $a_\lambda(x)$ in (3.1), then the bandwidth of this system will be $m = \max_\lambda = \{m_1, \dots, m_K\}$. The consequence is that the less smooth the variable coefficient functions are, the larger the bandwidth of the system. In the upcoming section, we demonstrate the application of the constructed operators and elucidate the solution process using an example.

3.2 Solving a boundary value problem

Consider the 1D boundary value problem (BVP) given by the Airy equation,

$$\varepsilon u''(x) - xu(x) = f(x), \tag{3.12}$$

on the interval $[-1, 1]$ and subject to the Dirichlet boundary conditions

$$u(-1) = Ai\left(-\varepsilon^{-1/3}\right), \qquad u(1) = Ai\left(\varepsilon^{-1/3}\right).$$

Here, $Ai(x)$ is the Airy function.[1] With $f(x) = 0$, the solution to (3.12) is given by $Ai(x\varepsilon^{-1/3})$. Discretising (3.12) involves employing the operators introduced to represent differentiation (3.3), multiplication by variable coefficients (3.8), and conversion between bases (3.11). This leads to the matrix and vector equation,

$$\Big(\varepsilon D_2 - S_1 S_0 M_0[x]\Big)\boldsymbol{u} = S_1 S_0 \boldsymbol{f}, \tag{3.13}$$

where $\boldsymbol{u}$ and $\boldsymbol{f}$ are vectors composed of Chebyshev expansion coefficients representing the functions $u(x)$ and $f(x)$, respectively. Importantly, the premultiplication of $\boldsymbol{f}$ by conversion matrices is a result of the basis change introduced by the differentiation operator on the left. To ensure the existence of a unique solution to this equation, the given boundary constraints of the problem must be imposed.

3.2.1 Implementing boundary conditions

The values of $T_k(x)$ and its first-order derivative $T_k'(x)$ at $x = \pm 1$ play a crucial role in defining Dirichlet and Neumann boundary conditions. They are expressed as follows:

$$T_k(\pm 1) = (\pm 1)^k, \qquad T_k'(\pm 1) = (\pm 1)^{k+1} k^2.$$

To enforce boundary constraints, a matrix that acts on the Chebyshev coefficients is required. For Dirichlet boundary conditions at $x = \pm 1$, such a matrix can be written as:

$$B_D = \begin{bmatrix} T_0(-1) & T_1(-1) & T_2(-1) & \dots \\ T_0(1) & T_1(1) & T_2(1) & \dots \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & \dots \\ 1 & 1 & 1 & \dots \end{bmatrix},$$

and for Neumann conditions at $x = \pm 1$ it will be:

$$B_N = \begin{bmatrix} T_0'(-1) & T_1'(-1) & T_2'(-1) & \dots \\ T_0'(1) & T_1'(1) & T_2'(1) & \dots \end{bmatrix} = \begin{bmatrix} 0 & 1 & -4 & 9 & \dots \\ 0 & 1 & 4 & 9 & \dots \end{bmatrix}.$$

[1] In MATLAB the Airy function is given by `airy`(x).

General boundary constraints, for example Robin conditions, can be encoded in a similar manner. In fact, any boundary condition that exhibits linear dependence on the solution's coefficients can be imposed in an automated manner [58]. It is important to assume that the boundary constraints are linearly independent; if not, at least one of them can be eliminated while preserving the uniqueness of the solution [73].

The matrices B_D and B_N operate on the coefficient vector $\boldsymbol{u}$ to return the values of the solution (or its derivative) on the boundaries. In the case of the Airy equation example, the ensuing linear system ensures that the solution evaluated at ± 1 satisfies the boundary conditions:

$$B_D \boldsymbol{u} = \begin{bmatrix} Ai(-\varepsilon^{-1/3}) \\ Ai(\varepsilon^{1/3}) \end{bmatrix}.$$

In general, one imposes K boundary conditions in the form $B\boldsymbol{u} = \boldsymbol{b}$ upon the linear system. By redefining the vector $\boldsymbol{f}$ in (3.13) to represent the coefficients of $f(x)$ within the $C^{(2)}$ basis and introducing $L = \varepsilon D_2 - S_1 S_0 M_0[x]$, one can reformulate eq. (3.13) as the sparse linear system $L\boldsymbol{u} = \boldsymbol{f}$. Combining the boundary conditions with those of the differential operator, one arrives at the infinite system of equations,

$$\begin{bmatrix} B \\ L \end{bmatrix} \boldsymbol{u} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{f} \end{bmatrix}. \tag{3.14}$$

An advantage of enforcing the boundary constraints in this manner is that the structure of the linear system is independent of the specific boundary constraints that are applied [58].

3.2.2 Solving the equation

Solving an infinite dimensional linear system is difficult. Olver and Townsend [58] suggest an adaptive QR approach which does so in a forward-solve like manner, terminating when the magnitude of the computer solution coefficients becomes sufficiently small. However, here we consider a simpler approach of taking a finite section approximation of (3.14), that is the first n columns is extracted from B, and from L the first n columns and $n - K$ rows are extracted.

Next, a matrix A is defined as a square operator of size $n \times n$, with its first K rows constituting the dense rows of the finite matrix B, and the remaining rows derived from the finite banded matrix L. Note that one could potentially have made this system overdetermined, but doing so would result in a more computationally expensive solution procedure without guaranteeing higher accuracy.

The vector $\boldsymbol{f}$ contains the first $n - K$ coefficients of $f(x)$ in the $C^{(2)}$ basis. In practice, one can approximate these coefficients by interpolating at $n - K$ Chebyshev points, then applying the Discrete Cosine Transform (DCT), and finally employing conversion operators to obtain coefficients in the $C^{(2)}$ basis. Similarly to A, the vector is redefined to include the boundary values $\boldsymbol{b}$ in its first K entries. This finally results in a sparse linear system $A\boldsymbol{u} = \boldsymbol{f}$.

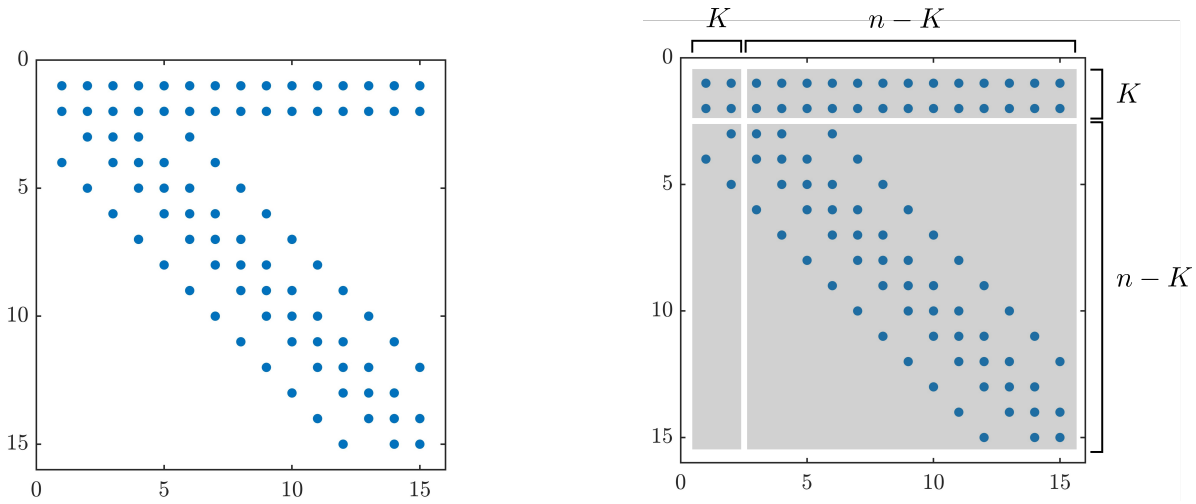

Figure 3.1: The left image depicts the distribution of nonzero entries in a finite section of the linear system (3.14). The matrix exhibits an almost-banded structure, that is, it is banded with the exception of the densely populated top rows, a result of incorporating the boundary constraints. Special techniques exist for solving this system efficiently. The figure on the right illustrates how such a matrix will be partitioned when using the Schur complement method for more efficient solves.

This system exhibits a distinctive *almost-banded* structure that is characteristic of linear systems within the ultraspherical method [23]. This structure is illustrated on the left in Figure 3.1. The K dense upper rows within the matrix can potentially hinder the computational efficiency of solving the linear system. Nevertheless, specific strategies have been devised to handle the inversion of such matrices while leveraging their inherent sparsity.

One effective technique known as the *Sherman–Morrison–Woodbury method* stands out in this context. It involves isolating the K densely-populated upper rows of the almost-banded matrix A from the remaining elements. This isolation is achieved by expressing A as a rank-K update of a banded matrix $\hat{A}$. That is $A = \hat{A} + UV^\top$, where both A and $\hat{A}$ assume square dimensions of size $n \times n$, and U and V are $n \times K$ matrices. Let I_K denote the identity matrix with dimensions $K \times K$ and assume that both $(I_K + V\hat{A}^{-1}U)$ and $\hat{A}$ are nonsingular, then the inverse of A can by calculated as [30]:

$$A^{-1} = \hat{A}^{-1} - \hat{A}^{-1}U(I_K + V^\top \hat{A}^{-1}U)^{-1}V^\top A^{-1}.$$

The advantage of this approach lies in the faster computation of the inverse of the banded matrix $\hat{A}$ when compared to the inverse of the almost-banded matrix A. If $\hat{A}$ has a bandwidth of m, the computational complexity for solving $\hat{A}^{-1}U$ would be in the order of $\mathcal{O}(m^2 n)$. Additionally, computing the term $(I_K + V^\top \hat{A}^{-1}U)^{-1}V^\top$ would introduce an additional complexity of approximately $\mathcal{O}(K^2 n)$ operations, resulting in an overall complexity in the order of $\mathcal{O}(m^2 n) + \mathcal{O}(K^2 n)$.

Another noteworthy approach to consider is the *Schur complement*, which is frequently employed in the solution of systems of linear equations. This method involves partitioning a matrix into four submatrices, a configuration that proves advantageous when the leading submatrix is relatively small, thus making the computation of its inverse efficient.

Suppose one partitions a square matrix A of dimensions $n \times n$ into four submatrices, each of which are nonsingular. Then the Schur complement of A can be expressed as [78]:

$$A^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} = \begin{bmatrix} S^{-1} & -S^{-1}FH^{-1} \\ -H^{-1}GS^{-1} & H^{-1} + H^{-1}GS^{-1}FH^{-1} \end{bmatrix},$$

where $S = E - GH^{-1}F$ is also a nonsingular matrix. In cases where A has K dense boundary rows at the top, it would be partitioned such that the leading submatrix E assumes dimensions $K \times K$. Then F will be $K \times (n-K)$, G is $(n-K) \times K$, and H is $(n-K) \times (n-K)$. This partitioning is visually depicted on the right in Figure 3.1. Given that the matrix H is banded with a bandwidth of m, determining $H^{-1}G$ requires $\mathcal{O}(m^2 n)$ operations. Moreover, assuming S has dimensions $K \times K$, solving $S^{-1}F$ would involve a computational complexity in the order of $\mathcal{O}(K^2 n)$. This is equivalent to the computational cost expected from the Sherman–Morrison–Woodbury formula.

Figure 3.2 on the left illustrates performance in terms of execution time of both strategies when employed to solve the linear system resulting from the Airy equation with $\varepsilon = 1$. As expected, they show comparable performance. It is clear that they significantly outperform the utilisation of MATLAB's backslash operator, which uses a sparse representation but does not exploit the system's almost-banded structure. Note that decreasing the value of ε significantly increases the number of terms n required to resolve the solution. However, it does not impact the bandwidth of the problem [58]. The approximate solution, computed through the Schur complement approach, is presented on the right in Figure 3.2, alongside the exact solution provided by MATLAB's Airy function, `airy`(x). The geometric convergence of this approximation is shown in Figure 3.3.

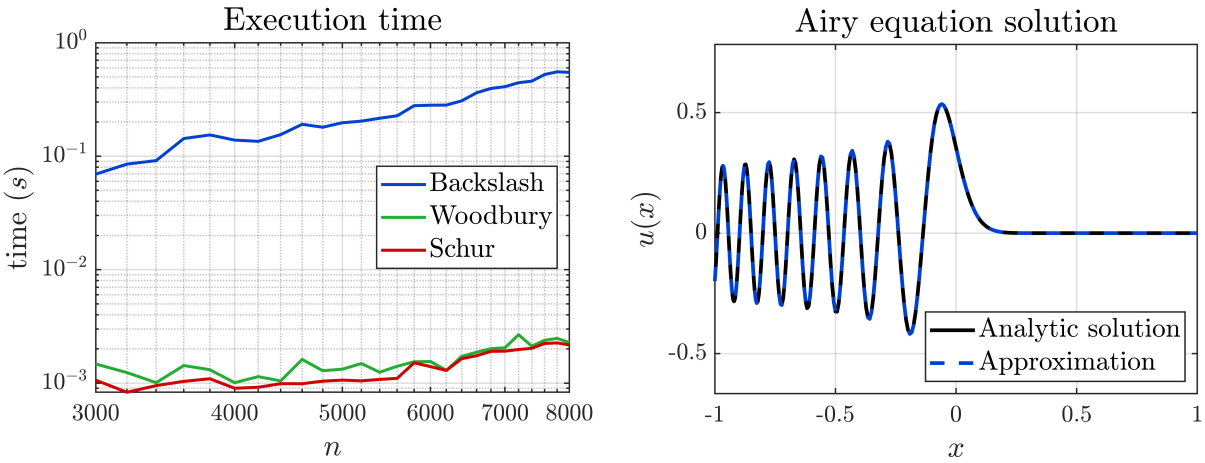

Figure 3.2: The left graph displays the execution time (in seconds) required for solving the almost-banded linear system (3.14) for increasing n using various approaches. In this case, these values of n are far larger than required to solve the ODE to machine precision (see Figure 3.3). However, these solution times are independent of ε, and as ε decreases, discretisations of this size are soon required. The Sherman–Morrison–Woodbury formula (green) and the Schur complement method (red) exhibit similar performance levels, both outperforming MATLAB's backslash operator (blue). On the right is the spectral solution to the Airy equation (3.12) with $\varepsilon = 0.0002$ depicted in blue, exhibiting a clear agreement with the analytic solution shown in black.

When addressing linear systems such as (3.14), another critical factor to consider is the condition number of the matrix A. This number provides insight into how close the matrix

is to singular and gives an indication of the forward accuracy to which the linear system can be solved. Generally, a large condition number indicates closeness to singularity and ill-conditioning of the linear system [9]. One way around this, is the use of *preconditioners*. Preconditioners serve as transformations that reconfigure a problem to a form that is more suitable to solve. The primary aim is to reduce the condition number of the problem.

The preconditioning process involves finding an approximate inverse P^{-1} of the original matrix A that satisfies two essential criteria. First, the matrix $P^{-1}A$ should have a reduced condition number compared to A. Second, solving linear systems involving $P^{-1}A$ should be computationally more efficient than solving systems directly with A.

Various preconditioning approaches are available, including diagonal, block-diagonal, and approximate inverse preconditioning [58]. The selection of a preconditioner in direct spectral methods is important, as the effectiveness of different techniques are usually based on the specific characteristics of the problem at hand.

In their work [58], Olver and Townsend demonstrate the existence of a diagonal preconditioner that ensures the preconditioned linear system resulting from the ultraspherical spectral method maintains a bounded condition number in the 2-norm. Specifically, for matrices with K boundary conditions (dense rows), they propose a diagonal preconditioner given by

$$P^{-1} = \frac{1}{2^{K-1}(K-1)!}\text{diag}\left(1, \ldots, \frac{1}{K}, \frac{1}{K+1}, \ldots\right). \tag{3.15}$$

The impact of preconditioning on the condition number of the linear system in (3.14) is illustrated in Figure 3.3, depicting a clear preservation of bounded condition numbers when preconditioning is used.

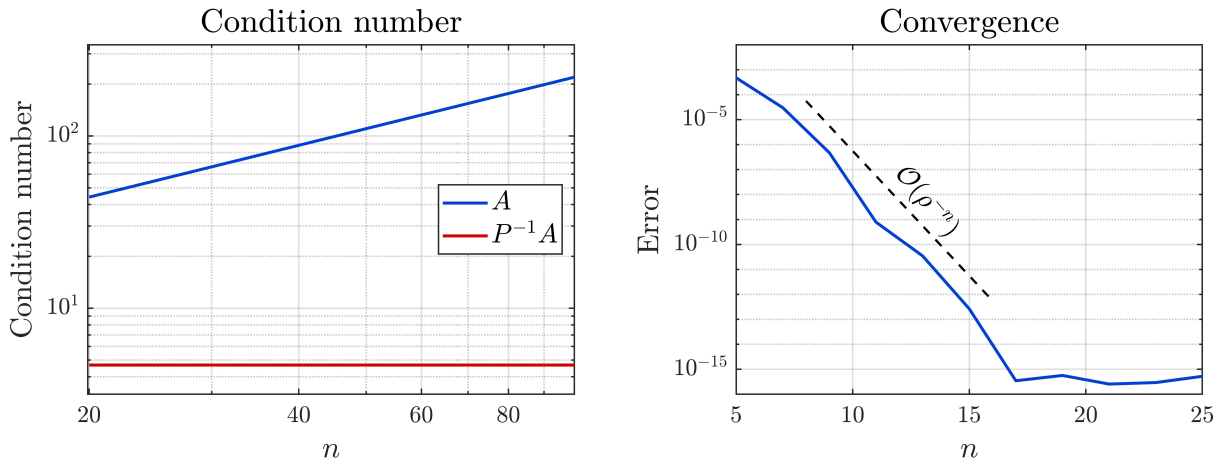

Figure 3.3: On the left side, the variation of the condition number of the system (3.14) with $\varepsilon = 0.01$ is depicted as n is increased. Observe that the application of the preconditioner in (3.15) appears to constrain the condition number at a constant value of 72.6. This behaviour was predicted by Olver and Townsend [58], and is in contrast to the rapid increase of the condition number when no preconditioning is employed. On the right, the geometric convergence with $\rho \approx 10$ of the solution to the Airy equation with $\varepsilon = 1$ is depicted. The roundoff plateau at roughly 10^{-16} is due to the accuracy reaching the level of machine precision. Fifteen digits of accuracy is attained when $n = 16$, however, if $\varepsilon = 0.0002$, then $n = 100$ is needed for the approximation to achieve the same accuracy. The value of ε impacts n, but does not affect the bandwidth of the linear system.

3.3 Time-dependent problems

The ultraspherical spectral method can be embedded in the method-of-lines type technique to offer an efficient extension for solving time-dependent problems [14]. To demonstrate, we consider solving the well-known heat equation on the interval $[0, L]$ over the time span $[0, T]$,

$$\frac{\partial}{\partial t}u(x,t) = c^2\frac{\partial^2}{\partial x^2}u(x,t), \tag{3.16}$$

with an initial condition $u(x,0) = (1+x^2)\,e^{-2x^2}$ and the Dirichlet boundary conditions $u(\pm 1) = 2e^{-2}$. This equation can model heat propagation in a uniform bar, where c represents a positive constant determined by material thermal properties like conductivity.

When addressing problems in the time domain, there are many available strategies, including high-order Runge–Kutta methods [12], exponential integrators [72], and fractional-step methods [44]. These approaches are typically categorised as explicit, implicit, or a combination of both. Explicit methods project the system's state into the future based on its present state, while implicit methods find solutions by solving equations involving both the current and future states.

Implicit methods have the additional computational cost of solving a linear or nonlinear system at each step and can present greater implementation challenges. Nevertheless, they prove significantly more efficient for solving stiff problems compared to explicit methods, since they are not constrained by severe time-step restrictions [38]. Defining precisely what constitutes a "stiff" equation is difficult, but it essentially implies that the equation contains terms that can lead to rapid variation in the solution. When explicit methods are employed for stiff problems, they often necessitate excessively small step sizes to uphold error limits and method stability. For such problems, achieving a desired level of accuracy with explicit methods demands considerably more computational time, making implicit methods with larger time steps a more efficient choice, even when factoring in the additional solve at each time step [4].

For linear time-dependent problems, we will employ an implicit, single step method named backward Euler, which is known to have an error of order one in time [9]. Backward Euler is chosen for its simplicity in both implementation and description, however the approach we describe could easily be extended to alternative implicit discretisations, such as Crank–Nicolson [17] A time step $\Delta t = 0.01$ is introduced, along with time points $t_n = n\Delta t$ for integers $n \geq 0$. The approximate solution at time t_i is denoted by $u^{[i]}$. Discretising in time using a finite difference approximation and treating all other terms implicitly results in equation (3.16) becoming a steady-state PDE in $u^{[i+1]}$,

$$\frac{u^{[i+1]} - u^{[i]}}{\Delta t} = c^2\frac{\partial^2 u^{[i+1]}}{\partial x^2}. \tag{3.17}$$

With implicit time-stepping stability is provided for all step sizes, avoiding the step-size limitations inherent to explicit methods. Rearranging (3.17) yields

$$\left(1 - c^2\Delta t\frac{\partial^2}{\partial x^2}\right)u^{[i+1]} = u^{[i]}.$$

Discretising in space using the spectral operators from Section 3.1 leads to:

$$\left(S_1 S_0 - c^2 \Delta t D_2\right) \boldsymbol{u}^{[i+1]} = S_1 S_0 \boldsymbol{u}^{[i]}, \tag{3.18}$$

where conversion matrices are necessary due to the basis change induced by the differentiation operator. Since the operators are all banded, the resulting matrix on the left also maintains a banded structure. Implicit time-stepping requires that this linear system be solved at each step in order to calculate the solution at the next time step. To initiate the scheme, we compute the Chebyshev coefficients of the initial condition using the DCT and set these as our starting solution, denoted as $\boldsymbol{u}^{[0]}$. We assume that the given initial condition is consistent with the boundary conditions, that is

$$B\boldsymbol{u}^{[0]} = \boldsymbol{b},$$

where B is the Dirichlet boundary condition matrix constructed in Section 3.2.1, and $\boldsymbol{b}$ contains the boundary values. When calculating the new solution at each step, one must ensure that the solution continues to satisfy the boundary constraints. To achieve this, the last two rows in the resulting matrix on the left of (3.18) is replaced with B, and the last two entries in the previous solution, denoted as $\boldsymbol{u}^{[i]}$, with $\boldsymbol{b}$. The inclusion of the dense rows in B results in the system exhibiting an almost-banded structure. Given that the matrix on the left remains constant across time steps, one can compute the LU decomposition for more efficient solving, particularly when dealing with numerous time steps. At each step, the righthand side must be updated to reflect the new solution, with the corresponding entries replaced by the boundary values. To visualise the solution at specific time steps, as shown on the left in Figure 3.4, the Chebyshev coefficients in the vector $\boldsymbol{u}$ can be transformed back into values at the Chebyshev points using the inverse DCT.

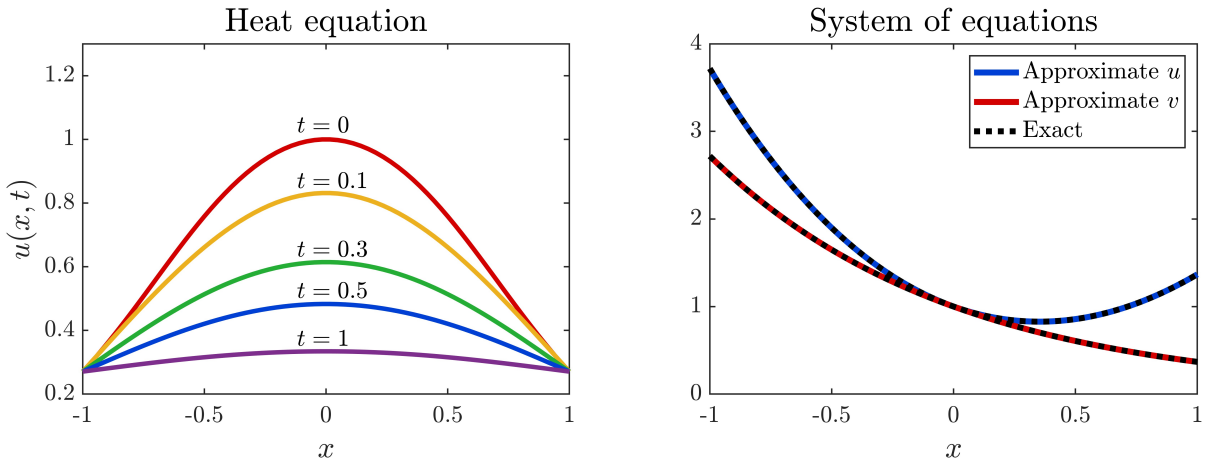

Figure 3.4: The left graph illustrates the spectral solution of the heat equation (3.16) with $c = 1$ at various time steps. The right graph depicts the solution to the set of coupled ODEs in (3.19). For both of these instances, the solution values at the Chebyshev points were computed from the Chebyshev expansion coefficients using the Discrete Cosine Transform (Section 2.3.3).

3.4 Systems of equations

The ultraspherical spectral method can be extended to handle systems of linear equations. We examine a simple set of coupled linear ODEs on the interval $[-1, 1]$ as an example:

$$\begin{aligned} u''(x) &= v(x) + f(x) \\ v''(x) &= u(x) + g(x), \end{aligned} \tag{3.19}$$

subject to Dirichlet boundary conditions. The approach to solving this system is almost equivalent to that of a single differential equation, albeit with the linear system's size being twice as large. Discretising both equations using the operators outlined in Section 3.1 and combining them into a single system, one obtains:

$$\begin{bmatrix} D_2 & -S_1S_0 \\ -S_1S_0 & D_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} S_1S_0\boldsymbol{f} \\ S_1S_0\boldsymbol{g} \end{bmatrix}, \tag{3.20}$$

where $\boldsymbol{u}$ and $\boldsymbol{v}$ contain the Chebyshev expansion coefficients of the solutions $u(x)$ and $v(x)$. Similarly, $\boldsymbol{f}$ and $\boldsymbol{g}$ denote the Chebyshev expansion coefficient vectors for approximating the functions $f(x)$ and $g(x)$, respectively.

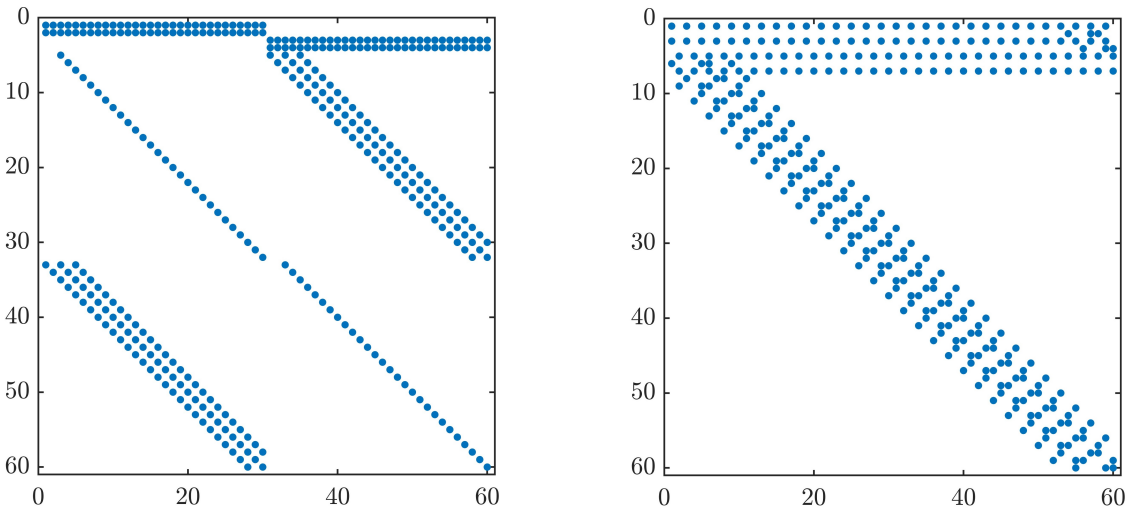

Figure 3.5: The distribution of nonzero entries in the linear system (3.19) before reordering (left) and after reordering (right) for $n = 60$. The fully and partially dense rows at the top of each of the matrices arise from enforcing the boundary constraints. Comparatively, the matrix on the right exhibits a significantly smaller bandwidth than the one on the left, signifying a more efficient structure for computational operations.

Incorporating boundary conditions similarly as before involves introducing four boundary rows that are partially dense, rather than two fully dense rows. Specifically, the last two rows in both the upper and lower halves of the matrix on the left in eq. (3.20) are removed. Following this, two partially dense rows are added at the top of the matrix to enforce the boundary conditions for u, succeeded by an additional two partially dense rows to impose boundary conditions for v. The arrangement of nonzero entries in the resulting matrix is illustrated on the left in Figure 3.5. While this matrix is visibly sparse, it lacks the almost-banded structure with a small bandwidth that was observed earlier. A reordering technique,

specifically interlacing the u and v coefficients, can improve the structure and restore the almost-banded shape, as evidenced on the right in Figure 3.5. During this procedure, the corresponding entries on the righthand side are substituted with the boundary values and the vector is appropriately reordered.

With this arrangement in place, the strategies from Section 3.2.2 can be employed to efficiently compute the matrix inversion to solve the linear system. Note that since the resulting solution has been reordered, it must be transformed back to its original configuration before extracting the initial n entries as coefficients of $\boldsymbol{u}$ and the latter n entries as coefficients of $\boldsymbol{v}$. These approximations for u and v are jointly plotted in Figure 3.4 for the case where $f(x) = 2$ and $g(x) = -x^2$, with $u(\pm 1) = 1 + e^{\pm 1}$ and $v(\pm 1) = e^{\pm 1}$ as boundary conditions.

The accuracy of the solution reaches roughly 10^{-15} when using 15 terms in the expansion. The approximate solution is compared to the exact solution given by $u(x) = e^{-x} + x^2$ and $v(x) = e^{-x}$. The accuracy is determined by calculating the 2-norm at 50 Chebyshev points.

3.5 Nonlinear problems

To demonstrate the applicability of the ultraspherical method to nonlinear problems, we consider a simple example on the interval $[-1, 1]$ defined by the following nonlinear differential equation:

$$u'' + xu^2 = 1, \qquad u(\pm 1) = \pm 1. \tag{3.21}$$

In this example, we employ Newton's method, a widely used iterative root-finding algorithm falling under the category of fixed-point iteration. Other examples of fixed-point iterations include Halley's method [8] and specific variations of Runge-Kutta methods [2].Newton's method proves successful in handling nonlinear problems by breaking them down into a series of linear problems. This involves first linearising the equation, and thereafter discretising it using spectral matrices. One can define an operator $\mathcal{F}$ to be a function of the solution u, such that

$$\mathcal{F}(u) = \mathcal{D}_2 u + \mathcal{M}(x)u^2 - \mathcal{I},$$

where $\mathcal{D}$, $\mathcal{M}$ and $\mathcal{I}$ are continuous operators that represent their counterparts in (3.21). The derivative of $\mathcal{F}$ with respect to u is known as the Fréchet derivative [65]. In the present context, it functions equivalently to a Jacobian with respect to u, and hence we will adhere to the more familiar terminology. The Jacobian function $\mathcal{J}$ is defined as

$$\mathcal{J}(u) = \frac{\partial \mathcal{F}(u)}{\partial u} = \mathcal{D}_2 + 2\mathcal{M}(x)u.$$

From this relationship one can define a small update in the solution, δu, as

$$\delta u = \mathcal{J}(u)^{-1}\mathcal{F}(u).$$

Next we discretise these operators using our spectral tools, leading to a discrete function for $\mathcal{F}$ in terms of the Chebyshev coefficient vector of $\boldsymbol{u}$,

$$F(\boldsymbol{u}) = D_2\boldsymbol{u} + S_1 S_0 M_0[\boldsymbol{x}] M_0[\boldsymbol{u}]\boldsymbol{u} - S_1 S_0 \boldsymbol{e}_1,$$

where $\boldsymbol{e}_1$ denotes first column of the identity matrix of size $n \times n$. The nonlinear term, u^2, is seen as a multiplication of u by itself, and can thus be discretised using a multiplication operator, as constructed in Section 3.1.2. Furthermore, conversion matrices are applied to ensure a consistent basis for solving. The Jacobian is discretised in a similar manner, leading to the matrix

$$J(\boldsymbol{u}) = D_2 + 2S_1 S_0 M_0[\boldsymbol{x}] M_0[\boldsymbol{u}].$$

We let $\boldsymbol{u}^{[i]}$ be the Chebyshev coefficients describing the solution at step i, which is updated every iteration using the formula:

$$\boldsymbol{u}^{[i+1]} = \boldsymbol{u}^{[i]} + \delta\boldsymbol{u}^{[i]},$$

where the update $\delta\boldsymbol{u}^{[i]}$ at step i is a vector determined by $\delta\boldsymbol{u}^{[i]} = J(\boldsymbol{u}^{[i]})^{-1} F(\boldsymbol{u}^{[i]})$. The process is initiated with an initial approximation, $\boldsymbol{u}^{[0]}$, and iteratively updated it to converge towards the accurate solution. The initial guess is selected to satisfy the boundary conditions. In this example, $u_0(x) = x$ is chosen as a suitable starting point, resulting in $\boldsymbol{u}^{[0]} = [0, 1, 0, \ldots]^\top$ as the initial Chebyshev coefficient vector.

To ensure that the update does not alter the boundary conditions, zero boundary conditions is imposed on the coefficients of the update. This is achieved through the boundary condition matrix B, by setting $B\delta\boldsymbol{u}^{[i]} = \boldsymbol{0}$ at each time step. This constraint is incorporated into the matrix J by replacing the last two rows, while zeros are substituted into the corresponding positions in F.

Four iterations of this strategy is applied while taking $n = 21$. This yields an approximation with roughly 13 digits of accuracy when compared to the solution generated by an open-source software system called Chebfun [63]. The approximation and solution is depicted in Figure 3.6, accompanied by the error after each iteration in Table 3.1.

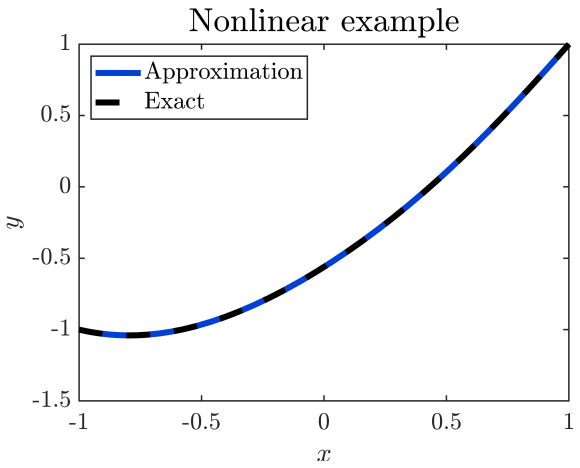

Figure 3.6: The approximate solution to (3.21) is showed to align with the exact solution.

Iteration i	2-norm error
1	4×10^{-4}
2	5×10^{-5}
3	6×10^{-11}
4	3×10^{-13}

Table 3.1: The 2-norm error after each iteration, measured at 50 Chebyshev points, is roughly quadratic.

3.6 Two-dimensional problems

Extending the ultraspherical spectral method to handle problems with higher spatial dimensions comes with increased computational demands, both in terms of storage and execution time [45]. However, it does not introduce significant additional conceptual complexity. To illustrate this, let us consider the extension of the method to solve a system of linear equations in two dimensions. The 2D analogue of the general form given in eq. (3.1) is expressed as:

$$\mathcal{L}u(x,y) = f(x,y), \qquad \mathcal{L} = \sum_{i=0}^{K_y}\sum_{j=0}^{K_x} a_{ij}(x,y)\frac{\partial^{i+j}}{\partial y^i \partial x^j}.$$

Here $\mathcal{L}$ represents a linear partial differential operator (PDO), K_x and K_y are the differential orders of $\mathcal{L}$ in the x- and y-variables respectively, and $f(x,y)$ and $a_{ij}(x,y)$ are given functions defined on $[-1,1]^2$. Additionally, standard linear constraints, such as Dirichlet, Neumann, or Robin boundary conditions, are often prescribed along the domain edges.

We seek to compute a matrix U of bivariate Chebyshev expansion coefficients of the solution $u(x,y)$, satisfying:

$$u(x,y) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} U_{ij}T_i(y)T_j(x), \qquad (x,y) \in [-1,1]^2. \tag{3.22}$$

3.6.1 The operators

Applying operators for derivative approximations and multiplications now require them to act in either the x- or y-direction. When addressing y-derivatives or multiplications involving y, the operator is applied to the *columns* of the coefficient matrix U, thereby necessitating premultiplication by the operator. Conversely, for x-derivatives or multiplications with x, the operator acts on the *rows* of U, which mandates postmultiplication by the transpose of the operator.

In instances where a variable coefficient $a(x,y)$ in the differential equation is separable, i.e., $a(x,y) = g(x)h(y)$, the discretisation of $a(x,y)u(x,y)$ can be managed by premultiplying U by $M_0[h]$ and postmultiplying by $M_0^\top[g]$. In cases where $a(x,y)$ is not separable, additional steps are required. The first is to find a low rank approximation to the function, that is,

$$a(x,y) \approx \sum_{r=1}^{N} g_r(x)h_r(y). \tag{3.23}$$

The open-source software package Chebfun2 [63] is a convenient tool for calculating low rank approximations of functions of two variables. It determines the approximation by means of an algorithm that can be viewed as an iterative application of Gaussian elimination with complete pivoting [74]. If $a(x,y)$ is approximated by the sum of several separable functions, as in (3.23), then the discretisation of $a(x,y)u(x,y)$ can be approximated by $\sum_{r=1}^{N} M_0[h_r]UM_0^\top[g_r]$.

It is sensible to ensure that when the Chebyshev polynomials dependent on x in eq. (3.22) are converted to ultraspherical polynomials, the Chebyshev polynomials dependent on y must be converted to the same basis. This requires premultiplication and postmultiplication by the conversion matrices (or their transposes), as computed in Section 3.1.3. To illustrate the application of these operators in 2D, we solve a Poisson-like equation with variable coefficients:

$$a_1(x,y)\frac{\partial^2 u}{\partial x^2} + a_2(x,y)\frac{\partial^2 u}{\partial y^2} = f(x,y). \tag{3.24}$$

For the sake of simplicity, we assume the coefficients are separable functions, i.e., $a_1(x,y) = g_1(x)h_1(y)$ and $a_2(x,y) = g_2(x)h_2(y)$, leading to the discrete form:

$$S_1 S_0 M_0[h_1] U D_2^\top M_2^\top[g_1] + M_2[h_2] D_2 U M_0^\top[g_2] S_0^\top S_1^\top = S_1 S_0 F S_0^\top S_1^\top, \tag{3.25}$$

where F is the expansion coefficient matrix of the function $f(x,y)$, calculated using the 2D DCT. This equation falls under the category of Sylvester equations [24], for which special methods exist to solve them efficiently [58]. However, these strategies are limited in the types of equations they can be applied to. Hence, a more general approach will be considered for solving (3.25).

To align with the 1D strategy, the matrix U is converted to a vector $\boldsymbol{u}$ of size $n^2 \times 1$ by stacking the columns from left to right. One can express this conversion of a matrix to a vector by $\boldsymbol{u} = \mathrm{vec}(U)$. Kronecker products are then utilised to enable the matrices in (3.25) to act on the vector $\boldsymbol{u}$. Setting $\boldsymbol{f} = \mathrm{vec}(S_1 S_0 F S_0^\top S_1^\top)$, and defining the discrete linear operator L to be the $n^2 \times n^2$ matrix:

$$L = \big(D_2^\top M_2^\top[g_1]\big) \otimes \big(S_1 S_0 M_0[h_1]\big) + \big(M_0^\top[g_2] S_0^\top S_1^\top\big) \otimes \big(M_2[h_2] D_2\big), \tag{3.26}$$

establishes the matrix and vector equation $L\boldsymbol{u} = \boldsymbol{f}$, as required. As previously discussed, all operators in this equation exhibit a banded structure, with the multiplication operator contributing the largest bandwidth. If m represents the highest polynomial degree required to approximate any of the variable coefficient functions, g_1, g_2, h_1, h_2, then this system would be block-banded with a bandwidth of $\mathcal{O}(mn)$. If the variable coefficients in the problem are smooth and can be approximated by polynomials of degree $m \ll n$, then the system would be sparse. In the upcoming section the boundary constraints are enforced in order to determine a nontrivial solution.

3.6.2 2D boundary conditions

The application of general boundary constraints in two dimensions is more complex than in one dimension [8]. Suppose the boundary conditions are supplied as linear constraints of the form $\mathcal{B}_x u(x,y) = \boldsymbol{g}(y)$ for left and right boundaries and $\mathcal{B}_y u(x,y) = \boldsymbol{h}(x)$ for bottom and top boundaries. One can discretise these conditions as

$$U B_x^\top = G^\top, \qquad B_y U = H,$$

where G is of size $K_x \times n$ and H has size $K_y \times n$, and they contain the first n Chebyshev coefficients of the functions in $\boldsymbol{g}(y)$ and $\boldsymbol{h}(x)$, respectively. Furthermore, B_x and B_y take on the same dimensions as G and H, respectively. If, for example, Dirichlet conditions are enforced on the four sides of the domain, then both B_x and B_y would be equal to the matrix B_D, constructed in Section 3.2.1. Note that for the constraints to be consistent, the matrices must satisfy the following *compatibility conditions* [73]:

$$HB_x^\top = (B_y U)B_x^\top = B_y(UB_x^\top) = B_y G^\top.$$

In the case of Dirichlet constraints on the reference square, the compatibility conditions are satisfied if the boundary data match at the four corners of $[-1,1]^2$. To incorporate the boundary conditions into the discrete operator in (3.26) two things are needed. Firstly, the conditions must be rewritten as Kronecker products to include them into the differential operator of size $n^2 \times n^2$. To do this, the new constraints are expressed as:

$$(B_x \otimes I)\,\boldsymbol{u} = \boldsymbol{g}, \qquad (I \otimes B_y)\,\boldsymbol{u} = \boldsymbol{h}, \tag{3.27}$$

where vectors $\boldsymbol{g}$ and $\boldsymbol{h}$ contain the stacked coefficients of G and H, respectively, resulting in a length of nK_x for $\boldsymbol{g}$ and nK_y for $\boldsymbol{h}$.

Secondly, the rows in L that correspond to the smallest coefficients in U must be removed in order for the system to remain square after incorporating the boundary constraints. The K_y bottom rows of U contains the smallest coefficients in the y-series, while the K_x rightmost columns contains the smallest coefficients in the x-series. If the entries of U are labelled 1 to n^2, down the columns and from left to right, and $\mathcal{I}_x$ denotes the indices that correspond to the smallest x-coefficients in U, then $\mathcal{I}_x = \{n^2 - nK_x + 1, \ldots, n^2\}$. Similarly, $\mathcal{I}_y = \{n - K_y + 1, \ldots, n, 2n - K_y + 1, \ldots, 2n, \ldots, n^2 - K_y + 1, \ldots, n^2\}$. The rows $\mathcal{I}_x \cap \mathcal{I}_y$ are removed from L and the corresponding entries are removed from $\boldsymbol{f}$.

Finally, the constraints expressed in (3.27) are appended at the top of L and at the top of $\boldsymbol{f}$. Once these conditions are incorporated, the linear system will have an almost block-banded structure, which can be solved in $\mathcal{O}(n^4)$ operations using one of the techniques described in Section 3.2.2, or an alternative, such as the adaptive QR method [68]. More advanced solvers also exist that can reduce this complexity to $\mathcal{O}(n^3)$ [58].

3.6.3 Example: Poisson's equation in 2D

Using the procedure outlined above, we solve for the constant coefficient Poisson equation $\nabla^2 u = -1$, with zero Dirichlet boundary conditions on $[-1,1]^2$. For comparison, we consider the Fourier series solution given by:

$$u(x,y) = \sum_{j=1}^{\infty}\sum_{k=1}^{\infty} \frac{f_{jk}}{(j^2+k^2)\pi^2}\sin(j\pi x)\sin(k\pi y),$$

where an explicit formula for f_{jk} can be computed by substituting in $f(x,y) = -1$

$$f_{jk} = 4\int_0^1\int_0^1 f(x,y)\sin(j\pi x)\sin(k\pi y)dxdy = \frac{16\left((-1)^j - 1\right)\left((-1)^k - 1\right)}{jk\pi^2}.$$

We regard this Fourier series, truncated at $j = k = 10^4$, as the 'exact' solution and compare our approximate solution to it at values on a grid of Chebyshev points. The 2-norm error is displayed in Figure 3.7. As anticipated for smooth solutions, fast convergence is observed as the polynomial degree n in the ultraspherical spectral approximation is increased. The accuracy reaches a plateau at roughly 10^{-11}. The solution is depicted on the left in the same figure.

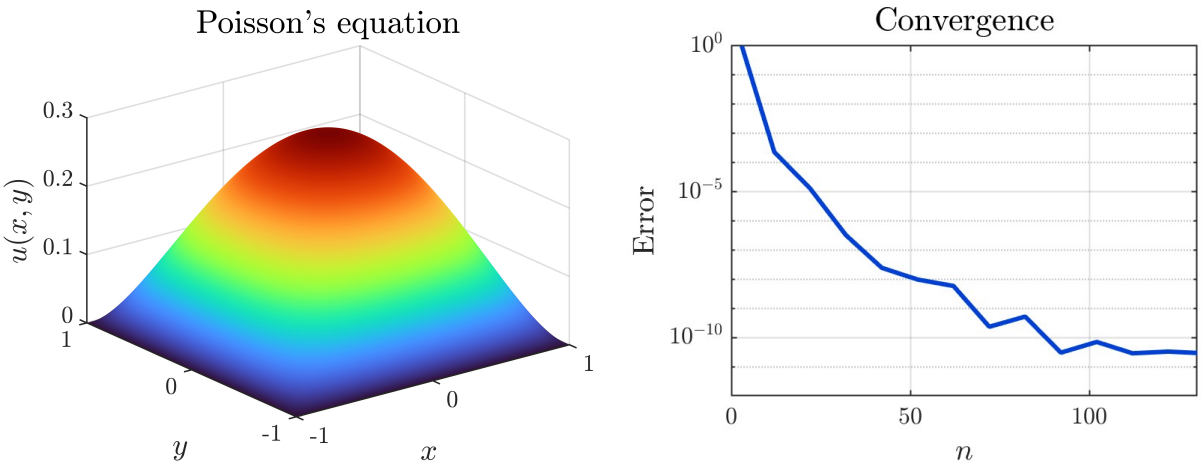

Figure 3.7: The left side illustrates the solution to the constant coefficient Poisson problem, $\nabla^2 u = -1$, under zero Dirichlet boundary conditions. On the right, the fast convergence of this solution is depicted. The error was computed as the 2-norm difference compared to the Fourier series solution at Chebyshev grid points.

CHAPTER 4

THE HPS METHOD

The spectral method explored thus far is known for its ease of implementation and impressive convergence properties. However, it comes with a limitation – it is primarily suitable for rectangular domains, thereby limiting its ability for solving problems on more irregular geometries. Over the last two decades, research on spectral methods has made significant strides, such as the introduction of the ultraspherical method [58], advancements in adaptive strategies [58], and the development of new techniques for unbounded domains [71] and time-dependent formulations [38]. Nevertheless, most optimal techniques are tailored for rectangular domains due to the challenge irregular geometries pose in terms of accuracy and efficiency [8]. New strategies for problem-solving on irregular domains are needed.

To overcome some of these challenges, domain decomposition emerges as a strategic solution, broadening the scope of domains where spectral methods can be effectively applied while breaking down the linear system into more manageable components [11]. An added advantage is the incorporation of parallelism facilitated by high-performance computing clusters, thus enhancing computational efficiency [23]. This approach also offers a means to optimally utilise computational resources and memory, particularly in the context of large-scale simulations.

Domain decomposition entails splitting a larger, potentially complicated, domain into smaller subdomains. The concept revolves around computing solutions for a given differential equation within each subdomain and subsequently recombining these solutions to derive a solution across the overarching domain. This transition signifies a shift from global spectral methods to the notion of a spectral element method. The approach allows for the solution of problems on more interesting geometries and, hopefully, a more resource-efficient computational process.

This chapter explores, in particular, a specific methodology known as the hierarchical Poincaré–Steklov method [4, 51, 52]. This strategy entails a multidomain spectral method grounded in a recursive domain decomposition approach. The essence of this technique lies in "gluing" solutions at the interfaces of elements through the application of Poincaré–Steklov operators. This hierarchical approach contributes to memory optimisation by segmenting the domain into smaller elements and employing localised operators, thus streamlining memory usage.

More specifically, we explore a variant of the hierarchical Poincaré–Steklov scheme that adopts the ultraspherical spectral method as an alternative to collocation for element discretisation, as introduced by Fortunato et al. [23]. This method was originally developed by Gillman and Martinsson [28, 29] for nodal discretisations, and so our initial focus is on explaining the fundamentals of the domain decomposition implementation within the modal framework.

4.1 Domain decomposition for modal discretisation

Several intricacies arise when tailoring the hierarchical domain decomposition approach introduced by Gillman and Martinsson to a modal discretisation. To facilitate our explanation, we consider a simple domain that has been partitioned into two subdomains, such as the 1D and 2D examples shown in Figure 4.1. When solving smooth PDEs on such domains, one can anticipate a smooth solution across the interface. For second order operators, it suffices to enforce that the global solution and its normal derivatives are continuous across the interface [29].

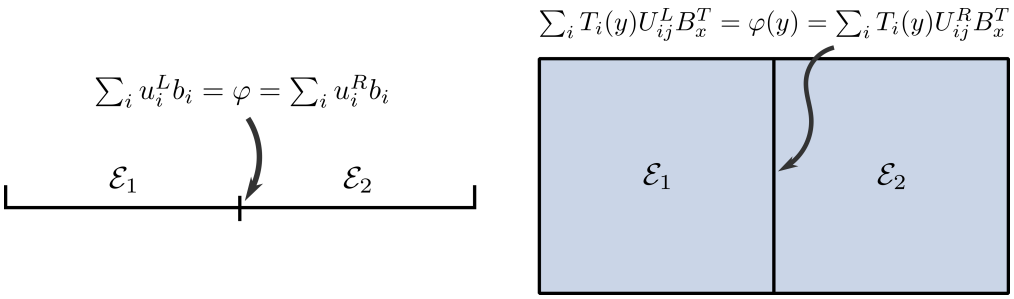

Figure 4.1: 1D and 2D examples of a domain partitioned into two subdomains, $\mathcal{E}_1$ and $\mathcal{E}_2$. In the modal setting, functions on each element are not directly communicating with each other, but rather interacting with an interface function on their shared boundary.

In the context of nodal discretisation, it is intuitive to categorise the nodes on each element into "interior" and "exterior" classes. At the interface where two elements meet, the values along that boundary are shared between the elements, making it straightforward to ensure that both elements yield the same solution at such points. Similarly, enforcing matching derivatives of the solution at the interface nodes is conceptually straightforward. However, in the modal setting, the coefficients within a Chebyshev expansion lack spatial positions, making it less intuitive to classify interface nodes. Instead, one needs to enforce that the bivariate Chebyshev expansions on each element yield identical solutions at their interfaces. To accomplish this, it is helpful to envision bivariate functions on each element not directly communicating with each other, but rather interacting with univariate functions at each interface [23], as illustrated in Figure 4.1. These univariate interface functions facilitate the classification of Chebyshev coefficients into similar groups as in nodal discretisation, allowing one to impose the joining conditions.

Constructing a linear system for a decomposed domain involves integrating three main components. The first component encompasses the equations that govern the solution within the domain, typically represented as a linear system $A\boldsymbol{u} = \boldsymbol{f}$. The second component is to incorporate linear boundary constraints in a similar manner to the global spectral method discussed in Chapter 3. Specifically, constraints on the exterior boundaries (excluding the interface boundary) are included as $B^e\boldsymbol{u} = \boldsymbol{b}$. The third component incorporates the joining conditions, crucial to the setup, acting on the interface boundary. These values are denoted by B^i and the derivatives by D^i. These components collectively form the linear system:

$$\begin{matrix}\text{Interior conditions}\\ \text{Boundary conditions}\\ \text{Joining conditions}\end{matrix}\left\{\begin{bmatrix} A_1 & & \\ & A_2 & \\ B_1^e & & \\ & B_2^e & \\ B_1^i & -B_2^i & \\ D_1^i & D_2^i & \end{bmatrix}\right. \begin{bmatrix}\boldsymbol{u}_1\\ \boldsymbol{u}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{f}_1\\ \boldsymbol{f}_2\\ \boldsymbol{b}_1\\ \boldsymbol{b}_2\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}.$$

Here, the subscripts indicate the index of the element it is applicable to. From this schematic, it is evident that the linear system could have been decoupled if not for the joining conditions. These conditions necessitate that the large linear system cannot be divided into smaller, independent parts for individual solving. The same principle applies to any number of subdomains. Every subdomain introduces interior, boundary, and joining conditions to the matrix depicted above, expanding its size significantly to $kn^2 \times kn^2$ for k subdomains. Solving such large systems efficiently poses a challenge, prompting us to explore a specific domain decomposition strategy that leverages a fast direct algorithm for efficient solves.

4.2 Outline of the HPS method

We introduce a specific domain decomposition strategy referred to in the literature as the hierarchical Poincaré–Steklov (HPS) approach, which we will apply to the spectral domain decomposition problem from the previous section. Our approach closely follows that introduced by Fortunato et al. [23] for modal discretisations and Gillman and Martinsson [28, 29] for nodal discretisations. The HPS strategy recursively joins together solutions at the interfaces between subdomains using Poincaré–Steklov operators. A broad outline of the method is:

1. A given domain is divided into a hierarchical tree of rectangular patches. The patches on the finest level are referred to as *leaf patches*, or simply *leaves*.

2. During the "initialisation stage" (Section 4.4.1) two local operators on each of the leaves are constructed – a *solution* operator, which computes the local solution to the PDE on the leaf when given Dirichlet data, and a *Dirichlet-to-Neumann* operator, which computes the outward flux of the local solution when given Dirichlet data.

3. The scheme then enters the "build stage" (Section 4.4.2), where local patches are merged pairwise to create parent patches in an upwards pass through the tree. Creating a parent patch consists of forming its solution operator and Dirichlet-to-Neumann operator by enforcing continuity of the solution and its derivative at the interface between its children. Merging continues until only a single global patch remains.

4. Once the solution operators for the entire hierarchy of patches has been created, we enter the final stage of the scheme, called the "solve stage" (Section 4.4.3). Here, we provide boundary conditions of the global domain to the global solution operator to retrieve the unknown interface data. This is repeated at every level of the tree in a downward pass through the tree. Ultimately, when the boundary data of the smallest patches are known, then it can be passed on to the local solution operators to solve the PDE.

To understand this method, we start by working through the simple domain decomposition setting of two square-shaped elements that are "glued" together.

4.3 Two "glued" squares

The domain decomposition setting with two square patches that are "glued" together is depicted in Figure 4.2. We wish to use the ultraspherical spectral method to solve on the domain $\Omega = [-2, 2] \times [-1, 1]$, which is decomposed into elements $\mathcal{E}_1 = [-2, 0] \times [-1, 1]$ and $\mathcal{E}_2 = [0, 2] \times [-1, 1]$. The problem setup is:

$$\begin{aligned} \nabla^2 u_1 &= f_1 && \text{in } \mathcal{E}_1, \\ \nabla^2 u_2 &= f_2 && \text{in } \mathcal{E}_2, \\ u_1 &= g_1 && \text{on } \partial\mathcal{E}_1 \cap \partial\Omega, \\ u_2 &= g_2 && \text{on } \partial\mathcal{E}_2 \cap \partial\Omega, \\ u_1 &= u_2 && \text{on } \Gamma, \\ \frac{\partial u_1}{\partial \boldsymbol{n}_1} &= \frac{\partial u_2}{\partial \boldsymbol{n}_2} && \text{on } \Gamma, \end{aligned} \tag{4.1}$$

where Γ is the interface between the two elements, f and g are given functions, and $f_i = f|_{\mathcal{E}_i}$ for any function f. The concept of a pairwise merge in this model problem plays a fundamental role within the framework of the HPS scheme.

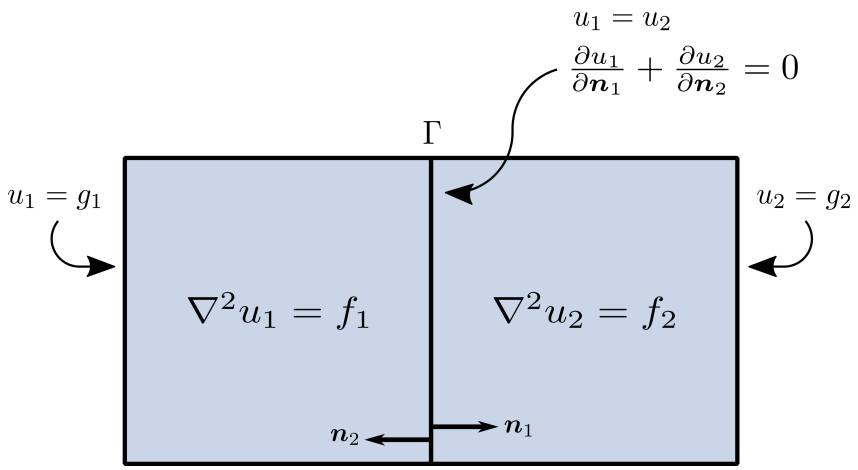

Figure 4.2: A problem setup with two "glued" patches, $\mathcal{E}_1$ and $\mathcal{E}_2$. The interface Γ is shared by the two elements. On Γ the solution on both patches must satisfy the joining conditions.

One can represent the solutions u_1 and u_2 on the two subdomains using $n \times n$ Chebyshev coefficient matrices.[1] As in our global spectral approach from Section 3.6, these matrices are compiled into vectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ of size $n^2 \times 1$. The functions f_1 and f_2 can be similarly discretised and converted into vectors, namely $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$. Figure 4.3 provides an illustration of the discrete version of the problem.

[1] In practice, the discretisation need not be the same on each patch, but we consider this case here for simplicity.

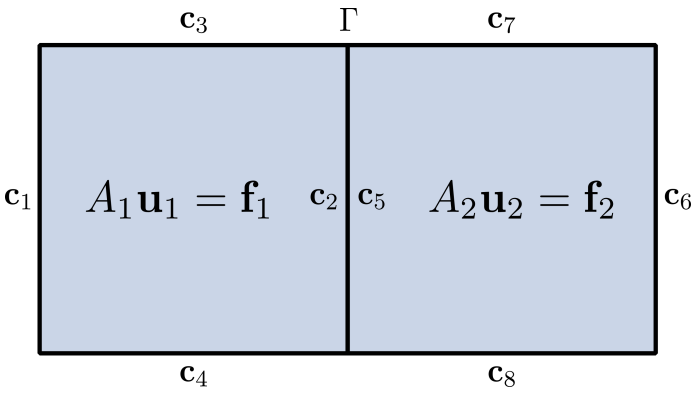

Figure 4.3: The discrete problem setup for the "glued" patches example. Vectors $\boldsymbol{c}_i$ contain the Chebyshev coefficients of the univariate function on side i.

In Figure 4.3, each vector $\boldsymbol{c}_i$ contains the n Chebyshev coefficients describing the univariate function of Dirichlet data on its corresponding side. Additionally, we create the vectors $\boldsymbol{c}_{\mathcal{E}_1}$ and $\boldsymbol{c}_{\mathcal{E}_2}$ to contain the Dirichlet data on the respective edges of each element, ordered as left, right, bottom, and top. That is:

$$\boldsymbol{c}_{\mathcal{E}_1} = \begin{bmatrix}\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4\end{bmatrix}^\top, \qquad \boldsymbol{c}_{\mathcal{E}_2} = \begin{bmatrix}\boldsymbol{c}_5, \boldsymbol{c}_6, \boldsymbol{c}_7, \boldsymbol{c}_8\end{bmatrix}^\top.$$

The vectors $\boldsymbol{c}_1, \boldsymbol{c}_3, \boldsymbol{c}_4, \boldsymbol{c}_6, \boldsymbol{c}_7$, and $\boldsymbol{c}_8$ are determined from the provided boundary conditions. Vectors $\boldsymbol{c}_2$ and $\boldsymbol{c}_5$, however, cannot yet be determined because they describe the unknown function on the interface Γ. To ensure continuity, one must have $\boldsymbol{c}_2 = \boldsymbol{c}_5 = \boldsymbol{c}^\Gamma$, where $\boldsymbol{c}^\Gamma$ is the Chebyshev coefficient vector describing the interface function on Γ.

Once $\boldsymbol{c}^\Gamma$ is determined, the two elements can be treated independently, and their solutions can be calculated separately. To calculate $\boldsymbol{c}^\Gamma$, we construct an interfacial solution operator S_Γ that takes in coefficients from the boundary of Ω. This can be expressed as:

$$\boldsymbol{c}^\Gamma = S_\Gamma \begin{bmatrix}\boldsymbol{c}_1, \boldsymbol{c}_3, \boldsymbol{c}_4, \boldsymbol{c}_6, \boldsymbol{c}_7, \boldsymbol{c}_8\end{bmatrix}^\top.$$

To compose such an operator on Ω, we first develop local direct solvers on $\mathcal{E}_1$ and $\mathcal{E}_2$. Components of these solvers are then utilised to construct the interfacial solution operator S^Γ. Once the interface function is determined, the two subproblems in (4.1) can be decoupled and independently solved by applying the local solvers. By using the direct solvers for the subproblems to build a direct solver for the global interface problem, it is clear that extending the strategy to multiple elements will follow readily.

4.3.1 Constructing the local operators

To establish a solver for (4.1), our initial step involves constructing operators that solve the local PDE within each of the elements $\mathcal{E}_1$ and $\mathcal{E}_2$. These operators, known as solution operators, are designed to process Dirichlet boundary data specific to each element and produce the corresponding PDE solution for that particular element. If the solution on an element $\mathcal{E}$ is represented using a $n \times n$ coefficient discretisation, then the dimensions of the corresponding solution operator, $S_\mathcal{E}$, will be $n^2 \times (4n + 1)$.

In the context of quadrilateral domains, the solution operator on $\mathcal{E}$ takes as input the four univariate functions representing the Dirichlet data along its four sides, which is given by $\boldsymbol{c}_\mathcal{E}$. The product $S_\mathcal{E}\boldsymbol{c}_\mathcal{E}$ yields the Chebyshev coefficients of the bivariate function that approximately solves the PDE on $\mathcal{E}$. On each element, $S_\mathcal{E}$ can be deconstructed into four distinct operators, namely $S^1_\mathcal{E}, S^2_\mathcal{E}, S^3_\mathcal{E}, S^4_\mathcal{E}$, each of size $n^2 \times n$, along with an additional column vector denoted as S^{rhs}. This decomposition is structured in such a way that:

$$S_\mathcal{E} = \left[\begin{array}{cccc|c} S^1_\mathcal{E} & S^2_\mathcal{E} & S^3_\mathcal{E} & S^4_\mathcal{E} & S^{\text{rhs}}_\mathcal{E} \end{array} \right].$$

On an element with boundary conditions described by vectors $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_4$, the solution is described as a superposition of these vectors:

$$\boldsymbol{u}_\mathcal{E} = S^1_\mathcal{E}\boldsymbol{c}_1 + S^2_\mathcal{E}\boldsymbol{c}_2 + S^3_\mathcal{E}\boldsymbol{c}_3 + S^4_\mathcal{E}\boldsymbol{c}_4 + S^{\text{rhs}}_\mathcal{E}.$$

If each of the boundary condition vectors are set to zero, i.e., $\boldsymbol{c}_1 = \ldots = \boldsymbol{c}_4 = \boldsymbol{0}$, then $\boldsymbol{u}_\mathcal{E} = S^{\text{rhs}}_\mathcal{E}$. This means the vector $S^{\text{rhs}}_\mathcal{E}$ is the solution to the PDE on $\mathcal{E}$ with homogeneous boundary conditions, i.e.,

$$\nabla^2 u^{\text{rhs}} = f|_\mathcal{E}, \qquad u^{\text{rhs}}|_{\partial\mathcal{E}} = 0,$$

and can therefore be seen as a particular solution to the problem. The vector $S^{\text{rhs}}_\mathcal{E}$ is thus defined by:

$$S^{\text{rhs}}_\mathcal{E} = \text{vec}(X), \qquad u^{\text{rhs}}(x,y) = \sum_{k=0}^{n-1}\sum_{\ell=0}^{n-1} X_{k\ell}\widehat{T}_k(y)\widehat{T}_\ell(x). \tag{4.2}$$

Despite it not being the conventional approach in most of the existing HPS research, we adopt the practice of including the particular solution within the solution operator. This choice is made to avoid repetitive descriptions of the linear algebra procedures when constructing the particular solution separately, and because it aligns with the implementation strategy outlined in the work by Fortunato et al. [23].

The last column of the local operator $S_\mathcal{E}$ was determined by setting homogeneous boundary conditions. Following a similar strategy to determine the individual components $S^i_\mathcal{E}$, we set a homogeneous righthand side, i.e., $f = 0$, and zero boundary conditions on all sides, except on side Φ_i of $\mathcal{E}$. Determining column j of this operator $S^i_\mathcal{E}$ requires solving:

$$\nabla^2 u^j = 0|_\mathcal{E}, \qquad u^j|_{\Phi_i} = \widehat{T}_{j-1}(x),$$

thereby defining each column of $S^i_\mathcal{E}$ as

$$\left(S^i_\mathcal{E}\right)_{:,j} = \text{vec}(X^j), \qquad u^j(x,y) = \sum_{k=0}^{n-1}\sum_{\ell=0}^{n-1} X^j_{k\ell}\widehat{T}_k(y)\widehat{T}_\ell(x).$$

Thus, the jth column of the solution operator associated with the ith side of element $\mathcal{E}$ is represented by $(S^i_\mathcal{E})_{:,j}$. Such a column is constructed by solving the homogeneous variant of the problem, with Dirichlet boundary conditions on all sides except on side Φ_i, on which the boundary condition is T_{j-1}.

Constructing the complete solution operator entails solving a total of $4n + 1$ problems, each sized $n^2 \times n^2$. Among these, $4n$ are equivalent homogeneous problems, differing only in their

boundary conditions. Consequently, efficient solving can be achieved using matrix factorisation techniques like LU factorisation or by storing the Schur complement factors in Sylvester form. If the problem can be formulated as a generalised Sylvester equation with a splitting rank of 2, the generalised Bartels–Stewart algorithm or the generalised Hessenberg–Schur algorithm [25] can be applied to solve each problem in $\mathcal{O}(n^3)$ operations. This approach results in an overall complexity of $\mathcal{O}(n^4)$ for solving n problems and constructing the operator $S_{\mathcal{E}}$. While this process needs to be repeated for each patch, the independence of the process for each patch allows for straightforward parallelisation.

It is worth noting that the Dirichlet data utilised in this construction process may exhibit discontinuities at the corners of the domain, introducing inconsistent boundary conditions. To guarantee compatibility, each function $\boldsymbol{c}_i$ is projected orthogonally onto the function space that preserves continuity at the corners prior to solving the PDE. For a more comprehensive explanation, refer to [23, Sec. 3.2.1].

At this stage, given the Dirichlet data on the boundaries, the solution for each individual element can be determined using their corresponding solution operators. In order to merge two elements or patches, the continuity conditions must be enforced. To achieve this, an operator is required to translate Dirichlet information into Neumann data, specifically along the boundaries of the elements. This operator, known as the Dirichlet-to-Neumann map on an element $\mathcal{E}$, denoted by $\Sigma_{\mathcal{E}}$, maps Dirichlet data on each side of $\mathcal{E}$ to the normal derivative or outward flux of the local solution to the PDE on each side of $\mathcal{E}$.

To construct such an operator, we consider how to calculate the normal derivatives on the edges of the reference square, i.e., at $x = \pm 1$ and $y = \pm 1$. As usual, the generalisation to arbitrary regular domains can be handled by affine transformations. The extension to more complex domains is discussed in Chapter 5. From the expansion of the solution u on this domain, discretised with the Chebyshev coefficient matrix U, we have:

$$\frac{\partial}{\partial x}u(\pm 1, y) = \sum_{i=0}^{n-1} T_i(y) \sum_{j=0}^{n-1} U_{ij} T_j'(\pm 1), \qquad \frac{\partial}{\partial y}u(x, \pm 1) = \sum_{j=0}^{n-1} T_j(x) \sum_{i=0}^{n-1} U_{ij} T_i'(\pm 1).$$

Furthermore, we introduce:

$$D_{\pm 1} = \pm \begin{bmatrix} T_0'(\pm 1) & T_1'(\pm 1) & \dots & T_{n-1}'(\pm 1) \end{bmatrix}, \qquad T_j'(\pm 1) = (\pm 1)^{j+1} j^2,$$

and define $\boldsymbol{d}_i$ to be the Chebyshev coefficient vector describing the normal derivative of the solution on side i. This enables the definition of the normal derivatives on the four sides as:

$$\begin{aligned} \text{left: } \boldsymbol{d}_1 &= U D_{-1}^{\top}, \\ \text{right: } \boldsymbol{d}_2 &= U D_{1}^{\top}, \\ \text{bottom: } \boldsymbol{d}_3 &= D_{-1} U, \\ \text{top: } \boldsymbol{d}_4 &= D_{1} U. \end{aligned}$$

Note the additional minus incorporated into the definition of $D_{\pm 1}$ to account for normal derivatives in the negative x- and y-directions. As is demonstrated in the 2D approach

detailed in Chapter 3, constraints are transformed to operators acting on the vector $\boldsymbol{u}$ through Kronecker products. The result is the normal derivative operator on the reference square,

$$D_{[-1,1]^2} = \begin{bmatrix} D_{-1} \otimes I \\ D_{+1} \otimes I \\ I \otimes D_{-1} \\ I \otimes D_{+1} \end{bmatrix},$$

where I is the $n \times n$ identity matrix. Such an operator must be applied to the coefficients of the solution and should thus be coupled with the solution operator of the element to produce the Dirichlet-to-Neumann map, $\Sigma_{\mathcal{E}} = D_{\mathcal{E}} S_{\mathcal{E}}$. As needed, the Dirichlet-to-Neumann map acts on the Dirichlet information on the boundaries and returns Neumann data:

$$\begin{bmatrix} \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ \boldsymbol{d}_3 \\ \boldsymbol{d}_4 \end{bmatrix} = \Sigma_{\mathcal{E}} \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \\ \boldsymbol{c}_3 \\ \boldsymbol{c}_4 \\ 1 \end{bmatrix}. \tag{4.3}$$

The scalar entry in the vector on the right is a result of including the particular solution as the last column of the solution operator.

4.3.2 Merging operators

Having constructed local operators on each element $\mathcal{E}_1$ and $\mathcal{E}_2$, the aim is now to build a global solution operator, S_Γ, from these local operators to solve for the coefficients of the unknown interface function, $\boldsymbol{c}^\Gamma$.

To do this, one partitions the boundary vectors in eq. (4.3) into "interior" and "interface" data. For an element $\mathcal{E}$, let $\boldsymbol{c}_{\mathcal{E}}^\Gamma$ and $\boldsymbol{d}_{\mathcal{E}}^\Gamma$ be vectors containing the coefficients describing the local Dirichlet and Neumann data, respectively, corresponding to the shared boundary Γ. Additionally, let $\boldsymbol{c}_{\mathcal{E}}^L$ and $\boldsymbol{d}_{\mathcal{E}}^L$ contain the coefficients describing the local Dirichlet and Neumann data corresponding to the unshared boundaries. For element $\mathcal{E}_1$, that has $\boldsymbol{c}_2$ on its interface, this would mean:

$$\begin{aligned} \boldsymbol{c}_{\mathcal{E}_1}^\Gamma &= \boldsymbol{c}_2, & \boldsymbol{c}_{\mathcal{E}_1}^L &= \begin{bmatrix} \boldsymbol{c}_1, \boldsymbol{c}_3, \boldsymbol{c}_4 \end{bmatrix}^\top, \\ \boldsymbol{d}_{\mathcal{E}_1}^\Gamma &= \boldsymbol{d}_2, & \boldsymbol{d}_{\mathcal{E}_1}^L &= \begin{bmatrix} \boldsymbol{d}_1, \boldsymbol{d}_3, \boldsymbol{d}_4 \end{bmatrix}^\top, \end{aligned}$$

and similarly, for element $\mathcal{E}_2$ with $\boldsymbol{c}_5$ on the interface:

$$\begin{aligned} \boldsymbol{c}_{\mathcal{E}_2}^\Gamma &= \boldsymbol{c}_5, & \boldsymbol{c}_{\mathcal{E}_2}^L &= \begin{bmatrix} \boldsymbol{c}_6, \boldsymbol{c}_7, \boldsymbol{c}_8 \end{bmatrix}^\top, \\ \boldsymbol{d}_{\mathcal{E}_2}^\Gamma &= \boldsymbol{d}_5, & \boldsymbol{d}_{\mathcal{E}_2}^L &= \begin{bmatrix} \boldsymbol{d}_6, \boldsymbol{d}_7, \boldsymbol{d}_8 \end{bmatrix}^\top. \end{aligned}$$

Based on the interaction between the Dirichlet-to-Neumann map and the Dirichlet data on Γ in eq. (4.3), the rows and columns of the local operators $\Sigma_{\mathcal{E}_1}$ and $\Sigma_{\mathcal{E}_2}$ can be partitioned into "interior" and "interface" blocks using the same notation.

This gives the partitioned version of eq. (4.3) on $\mathcal{E}_1$ as:

$$\begin{bmatrix} \boldsymbol{d}^L_{\mathcal{E}_1} \\ \boldsymbol{d}^\Gamma_{\mathcal{E}_1} \end{bmatrix} = \left[\begin{array}{cc|c} \Sigma^{L,L}_{\mathcal{E}_1} & \Sigma^{L,\Gamma}_{\mathcal{E}_1} & \Sigma^{L,\text{end}}_{\mathcal{E}} \\ \Sigma^{\Gamma,L}_{\mathcal{E}_1} & \Sigma^{\Gamma,\Gamma}_{\mathcal{E}} & \Sigma^{\Gamma,\text{end}}_{\mathcal{E}} \end{array}\right] \begin{bmatrix} \boldsymbol{c}^L_{\mathcal{E}_1} \\ \boldsymbol{c}^\Gamma_{\mathcal{E}_1} \\ 1 \end{bmatrix}, \tag{4.4}$$

and similarly on $\mathcal{E}_2$ as:

$$\begin{bmatrix} \boldsymbol{d}^L_{\mathcal{E}_2} \\ \boldsymbol{d}^\Gamma_{\mathcal{E}_2} \end{bmatrix} = \left[\begin{array}{cc|c} \Sigma^{L,L}_{\mathcal{E}_2} & \Sigma^{L,\Gamma}_{\mathcal{E}_2} & \Sigma^{L,\text{end}}_{\mathcal{E}_2} \\ \Sigma^{\Gamma,L}_{\mathcal{E}_2} & \Sigma^{\Gamma,\Gamma}_{\mathcal{E}_2} & \Sigma^{\Gamma,\text{end}}_{\mathcal{E}_2} \end{array}\right] \begin{bmatrix} \boldsymbol{c}^L_{\mathcal{E}_2} \\ \boldsymbol{c}^\Gamma_{\mathcal{E}_2} \\ 1 \end{bmatrix}. \tag{4.5}$$

Here, superscripts denote row and column indices for slicing a matrix or vector and "end" denotes the index of the last column of a matrix. The last column of the matrices in (4.4) and (4.5) encode the contribution from the particular solution. For the continuity condition to be satisfied, the derivatives at the interface must be equal on both patches, thus $\boldsymbol{d}^\Gamma_{\mathcal{E}_1} = \boldsymbol{d}^\Gamma_{\mathcal{E}_2}$. Setting these parts of the above equations equal, yields:

$$-\left[\begin{array}{cc|c} \Sigma^{\Gamma,L}_{\mathcal{E}_1} & \Sigma^{\Gamma,\Gamma}_{\mathcal{E}_1} & \Sigma^{\Gamma,\text{end}}_{\mathcal{E}_1} \end{array}\right] \begin{bmatrix} \boldsymbol{c}^L_{\mathcal{E}_1} \\ \boldsymbol{c}^\Gamma_{\mathcal{E}_1} \\ 1 \end{bmatrix} = \left[\begin{array}{cc|c} \Sigma^{\Gamma,L}_{\mathcal{E}_2} & \Sigma^{\Gamma,\Gamma}_{\mathcal{E}_2} & \Sigma^{\Gamma,\text{end}}_{\mathcal{E}_2} \end{array}\right] \begin{bmatrix} \boldsymbol{c}^L_{\mathcal{E}_2} \\ \boldsymbol{c}^\Gamma_{\mathcal{E}_2} \\ 1 \end{bmatrix},$$

and isolating the unknown interface coefficients $\boldsymbol{c}^\Gamma$ gives:

$$-\left(\Sigma^{\Gamma,\Gamma}_{\mathcal{E}_1} + \Sigma^{\Gamma,\Gamma}_{\mathcal{E}_2}\right)\boldsymbol{c}^\Gamma = \left[\begin{array}{cc|c} \Sigma^{\Gamma,L}_{\mathcal{E}_1} & \Sigma^{\Gamma,L}_{\mathcal{E}_2} & \Sigma^{\Gamma,\text{end}}_{\mathcal{E}_1} + \Sigma^{\Gamma,\text{end}}_{\mathcal{E}_2} \end{array}\right] \begin{bmatrix} \boldsymbol{c}^L_{\mathcal{E}_1} \\ \boldsymbol{c}^L_{\mathcal{E}_2} \\ 1 \end{bmatrix}.$$

This means the interfacial solution operator can be defined by:

$$S^\Gamma = -\left(\Sigma^{\Gamma,\Gamma}_{\mathcal{E}_1} + \Sigma^{\Gamma,\Gamma}_{\mathcal{E}_2}\right)^{-1}\left[\begin{array}{cc|c} \Sigma^{\Gamma,L}_{\mathcal{E}_1} & \Sigma^{\Gamma,L}_{\mathcal{E}_2} & \Sigma^{\Gamma,\text{end}}_{\mathcal{E}_1} + \Sigma^{\Gamma,\text{end}}_{\mathcal{E}_2} \end{array}\right],$$

and will produce the n Chebyshev coefficients of the solution to the PDE on the interface Γ when given the Dirichlet data on the boundary of the merged domain, that is:

$$\boldsymbol{c}^\Gamma = S^\Gamma \begin{bmatrix} \boldsymbol{c}^L_{\mathcal{E}_1} \\ \boldsymbol{c}^L_{\mathcal{E}_2} \\ 1 \end{bmatrix}. \tag{4.6}$$

Using this, one can construct a new Dirichlet-to-Neumann operator on the global domain, Σ_Ω. Since (4.4) and (4.5) already provide equations for $\boldsymbol{d}^L_{\mathcal{E}_1}$ and $\boldsymbol{d}^L_{\mathcal{E}_2}$, one can extract these parts and separate the terms containing $\boldsymbol{c}^\Gamma$:

$$\begin{bmatrix} \boldsymbol{d}^L_{\mathcal{E}_1} \\ \boldsymbol{d}^L_{\mathcal{E}_2} \end{bmatrix} = \left[\begin{array}{cc|c} \Sigma^{L,L}_{\mathcal{E}_1} & 0 & \Sigma^{L,\text{end}}_{\mathcal{E}_1} \\ 0 & \Sigma^{\Gamma,L}_{\mathcal{E}_2} & \Sigma^{\Gamma,\text{end}}_{\mathcal{E}_2} \end{array}\right] \begin{bmatrix} \boldsymbol{c}^L_{\mathcal{E}_1} \\ \boldsymbol{c}^L_{\mathcal{E}_2} \\ 1 \end{bmatrix} + \begin{bmatrix} \Sigma^{L,\Gamma}_{\mathcal{E}_1} \\ \Sigma^{\Gamma,\Gamma}_{\mathcal{E}_2} \end{bmatrix} \boldsymbol{c}^\Gamma.$$

Substituting (4.6) into this equation gives the global Dirichlet-to-Neumann map on Ω as:

$$\Sigma_\Omega = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_1}^{L,L} & 0 & \Sigma_{\mathcal{E}_1}^{L,\text{end}} \\ 0 & \Sigma_{\mathcal{E}_2}^{\Gamma,L} & \Sigma_{\mathcal{E}_2}^{\Gamma,\text{end}} \end{array}\right] + \begin{bmatrix} \Sigma_{\mathcal{E}_1}^{L,\Gamma} \\ \Sigma_{\mathcal{E}_2}^{\Gamma,\Gamma} \end{bmatrix} S^\Gamma,$$

which will produce a vector representing the normal derivatives of the solution to the PDE on the six edges of Ω when given the Dirichlet data on these edges:

$$\begin{bmatrix} \boldsymbol{d}_{\mathcal{E}_1}^L \\ \boldsymbol{d}_{\mathcal{E}_2}^L \end{bmatrix} = \Sigma_\Omega \begin{bmatrix} \boldsymbol{c}_{\mathcal{E}_1}^L \\ \boldsymbol{c}_{\mathcal{E}_2}^L \\ 1 \end{bmatrix}. \tag{4.7}$$

The key takeaway from this section is that one can leverage the Dirichlet-to-Neumann operators of two patches to eliminate their shared interface data and create a merged solution operator for their parent patch, which in turn is used to create a Dirichlet-to-Neumann operator for the parent patch.

4.3.3 Computing the solution

With all the needed ingredients to compute the solution to (4.1), we start by solving for the interface function on Γ. That is, we compute the n Chebyshev coefficients in the vector $\boldsymbol{c}^\Gamma$ via the matrix-vector product involving the global solution operator in (4.6). Now that the Dirichlet data is established on all four sides of each element $\mathcal{E}_1$ and $\mathcal{E}_2$, the application of local solution operators, $S_{\mathcal{E}_1}$ and $S_{\mathcal{E}_2}$, is possible. These operators return the $n^2 \times 1$ coefficient vectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$, which can be reordered to $n \times n$ coefficient matrices and used to approximate the solutions u_1 and u_2, respectively, satisfying (4.1).

4.4 The hierarchical scheme

The model problem of Section 4.3 showed how to construct local operators on two elements, $\mathcal{E}_1$ and $\mathcal{E}_2$, and how to "glue" these operators together to provide similar operators on the global domain Ω. The merged solution operator, S_Γ, solves for the unknown interface inside Ω, and the merged Dirichlet-to-Neumann operator, Σ_Ω, maps boundary data to outward fluxes on Ω. These operators encapsulate the needed knowledge to solve the PDE on Ω. Essentially, Ω now resembles the original elements $\mathcal{E}_1$ or $\mathcal{E}_2$, thereby enabling the treatment of it as yet another element that can be merged again with a new domain. This will be the case after every merge – the resulting domain can be seen as a self-contained unit with access to its local operators. This framework embodies the HPS scheme.

The computational complexity for constructing the solution operator on an element is calculated as $\mathcal{O}(n^4)$. Similarly, the Dirichlet-to-Neumann operator on an element can be computed as a matrix product in $\mathcal{O}(n^4)$ operations [23]. These estimations aid in determining the computational complexity of the HPS method. We will now proceed to outline the complexity for each of the three stages that encapsulate the HPS method, starting with the initialisation stage.

4.4.1 The initialisation stage

The HPS scheme commences with the initialisation phase, during which a global domain Ω is decomposed into a mesh of M elements, $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{M}$. The example in Figure 4.4 illustrates the decomposition into rectangular elements if $M = 4$. Subsequently, local solution operators and Dirichlet-to-Neumann maps are constructed for each element $\mathcal{E}_i$, according to the guidelines provided in Section 4.3.1. Since the steps executed at this stage pertain exclusively to each element, the algorithm performing these operations can be executed in parallel across the various elements.

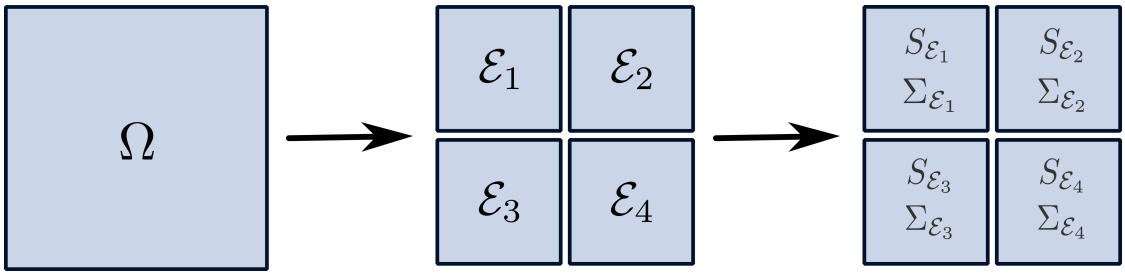

Figure 4.4: During the initialisation stage of the HPS method, a domain is divided into a number of subdomains, called leaves. On each of the leaves, a solution operator and a Dirichlet-to-Neumann operator is constructed.

As the solution and Dirichlet-to-Neumann operators are each computed once for every element, the overall computational cost of the initialisation stage will scale as:

$$Mn^4 \approx Nn^2,$$

where $N \approx Mn^2$ is the number of degrees of freedom. This is calculated as the product of the number of coefficients describing each patch and the total number of patches.

4.4.2 The build stage

Once the initialisation of local operators has been accomplished for each leaf, the methodology proceeds to the build stage. Typically, the mesh of elements is accompanied by a set of indices that detail the order of merging. Such a sequence comprises a series of pairwise merging instructions starting with the leaves and subsequently encompassing parent patches. Adhering to this order, we traverse through the hierarchical structure, starting at the leaves and building up to the global domain.[2] Along the way, the merged parent patches retain their newly calculated solution operators and Dirichlet-to-Neumann mappings, as described in Section 4.3.2. This procedure is graphically depicted in Figure 4.5.

[2]The concept of 'leaves' is standard in domain decomposition terminology. However, since the 'tree' is usually as depicted in Figure 4.5, they might be better named 'roots'.

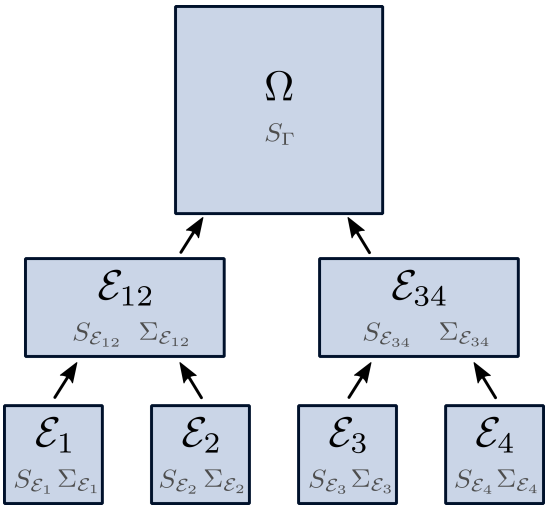

Figure 4.5: During the build stage of the HPS method, local patches are merged pairwise in an upwards pass though the tree. This merging involves using the local operators of the children to construct a solution operator and Dirichlet-to-Neumann operator for the parent patch. Merging continues until only the global domain remains.

The computational cost of the build stage [23, Sec. 3.4] scales as

$$n^3 M^{3/2} \approx N^{3/2},$$

and results in the construction of a global solution operator capable of operating on the entire mesh. It takes in Dirichlet data from every boundary of Ω and in return provides the solution to the PDE along the merged interface of the highest level of the hierarchy.

4.4.3 The solve stage

The final phase within the HPS scheme is the solve stage. During this stage, the merged solution operators are used to calculate the unknown interface data at each level in a downwards traversal through the hierarchy. The initial step involves employing the global solution operator constructed in the build stage on Dirichlet data from every boundary of the global mesh.

If Neumann data is provided on the boundaries instead of Dirichlet conditions, it is necessary to translate this information into Dirichlet conditions before applying the global solution operator. Fortunately, we have devised the global Dirichlet-to-Neumann map, which can be inverted to convert Neumann data into Dirichlet data. Once this conversion is accomplished, the global solution operator can be effectively applied and the subsequent process remains identical to when Dirichlet conditions were imposed.

The global solution operator will return the Chebyshev coefficients representing the solution at the merged interface of the highest level of the hierarchy. These coefficients then serve as the basis for Dirichlet data on the subsequent level to be used once again with solution operators to calculate the unknown interface data for the next level of subdomains. Eventually, the lowest tier is reached, which consists of the leaves. At this point, the solution is known at

every interface connecting the elements and the localised solution operators $S_{\mathcal{E}_i}$ can be put into action to compute the bivariate solution in the interior of each element $\mathcal{E}_i$. This procedure is depicted in Figure 4.6.

The computational cost of the solve stage [23, Sec. 3.4] scales as:

$$n^2 M \log M + n^3 M \approx N \log M + Nn,$$

giving the overall complexity of the method as:

$$\underbrace{Nn^2 + N^{3/2}}_{\text{initialisation \& build}} + \underbrace{N \log M + Nn}_{\text{solve}} \approx Nn^2 + N^{3/2}. \tag{4.8}$$

For comparison, the HPS scheme based on spectral collocation typically achieves an overall complexity of $\mathcal{O}(Nn^4 + N^{3/2})$ [52], making (4.8) a notable improvement, particularly in the high n regime. It is clear that the majority of the computational work is done in the initialisation and build stages. Since the boundary conditions of the problem are only imposed in the solve stage, solving the same problem multiple times with varying boundary data becomes cost-effective, as it only requires re-executing the solve stage [52].

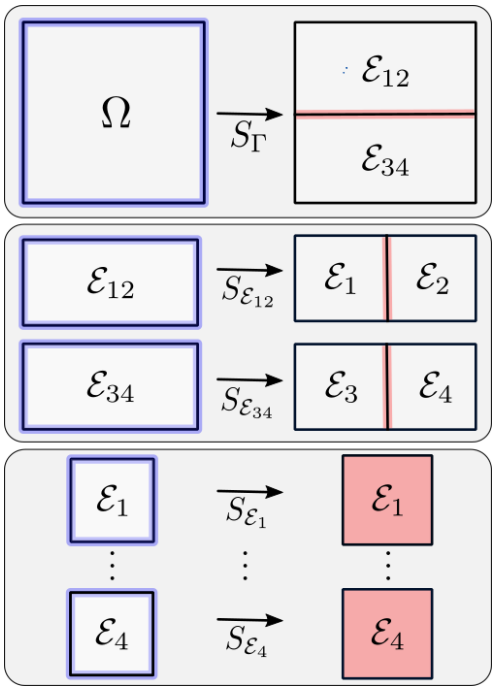

Figure 4.6: During the solve stage of the HPS method, the boundary conditions are provided to the global solution operator to retrieve unknown interface data. This is repeated at every level of the tree in a downward pass. At the lowest level, the local solution operators of the leaves are employed to calculate the solutions at the interior of the subdomains.

An effective strategy also exists for solving the same problem but with different righthand sides f. Recall that all the columns in the local solution operator, except the last one, were determined by solving homogeneous problems. The righthand side only contributed to calculating this last column by solving eq. (4.2). Thus, if the righthand side changes, it is only this column in the solution and Dirichlet-to-Neumann operators of the leaves that

requires updating. Instead of solving $4n + 1$ problems to construct a new solution operator, one only needs to solve a single problem to update the operator, resulting in a reduced complexity of $\mathcal{O}(Nn)$ for the initialisation stage. To carry this update through the entire hierarchy, a modified build stage must be conducted. In this process, the last column of the interfacial solution and Dirichlet-to-Neumann operators are updated instead of recalculating them completely. The modified build stage would have complexity $\mathcal{O}(NM^{1/2})$, giving the overall complexity as $\mathcal{O}(Nn + NM^{1/2})$. This strategy is particularly beneficial when implicit time-stepping schemes are applied and numerous iterations with changing righthand sides need to be calculated [5].

4.5 Refinement

In FEM and SEM, there are two main types of refinement used to improve the accuracy of a discretisation: h-refinement and p-refinement. These refinements can be uniform or global, meaning they are applied in the same way across the entire domain, or they can be local, meaning they are applied only to a small number of elements. Applying p-refinement to an element involves increasing the polynomial degree,[3] while h-refinement would involve partitioning the domain into smaller subdomains (with the same polynomial degree as the original element). In this context, h refers to the minimum average element width in the discretisation.

Typically, to increase accuracy in the vicinity of complex geometries, such as reentrant corners, local refinement applied to the surrounding elements will provide the best results [42]. Adaptive refinement strategies exist that automates the mesh or polynomial degree adjustments based on error criteria, optimizing computational resources to achieve accuracy with minimal user involvement. Geldermans and Gillman present one such scheme [26], however our focus here is on manual refinement, reliant on user-guided decisions.

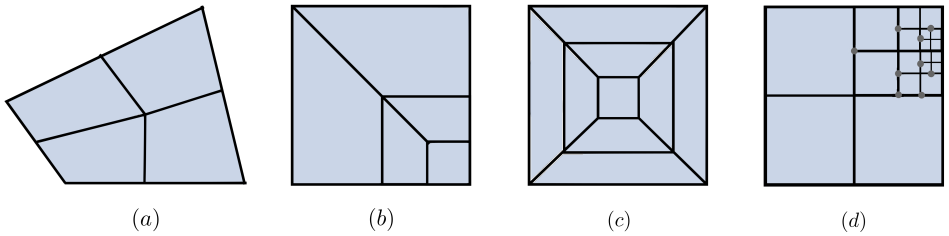

Figure 4.7: Image (a) depicts an example of uniform refinement of a quadrilateral, which connects the midpoint of each edge to the centre of the shape to produce four quadrilateral subdomains. In (b) and (c) non-uniform/local refinement strategies are presented, as suggested by [23], for the refinement into corners and around points, respectively. These schemes aim to avoid hanging nodes. Image (d) depicts a local h-refinement strategy resulting in hanging nodes.

Local h-refinement of a domain risks creating nodes in the mesh that occur in the middle of an element boundary. Such points are called "hanging nodes", and are demonstrated in image (d) in Figure 4.7. While hanging nodes may be handled in the HPS scheme through the use

[3] In FEM/SEM the polynomial degree of an element is usually indicated by p, whereas in spectral methods n is typically used (as in Chapter 3).

of interpolation operators [26], it is simpler to design a local refinement scheme that avoids such occurrences. Examples of such schemes, suggested in [23], are demonstrated in image (b) in Figure 4.7 for local refinement into corners, and image (c) for refinement around a point.

Applying uniform h-refinement results in scaling the number of patches in the mesh as $M = \mathcal{O}(2^{2r})$, where r represents the refinement level. The refinement level start at $r = 0$ signifying no domain refinement and then advances with $r = 1$ as the domain is partitioned into four subdomains. Image (a) in Figure 4.7 demonstrates one level of refinement of a quadrilateral domain. This example shows a possible global refinement approach for non-rectangular elements. It connects the centre of the element to the midpoints on each of the edges, resulting in four new quadrilaterals as subdomains.

4.6 Software

The ultraspherical spectral element method discussed in this chapter was originally introduced by Fortunato et al. [23] and has been implemented as an open-source software system named `ultraSEM` [22]. This software closely adheres to the steps of the HPS method and provides a user-friendly platform for performing spectral element computations within MATLAB. Here, we provide a brief overview of how this software can be utilised.

Users create a domain using the `ultraSEM.Domain` class, which encodes information about the domain shape and position. The software has several standard domains built-in, including rectangles, quadrilaterals, and triangles. Rectangular domains are dealt with via the process described above. Quadrilaterals and triangles require the addition of appropriate mappings, which we discuss in the following chapter.

It is possible to merge multiple domains (of either the same or differing types) to form larger domains using the '`&`' operator in MATLAB. During this process, merge indices are saved to reflect the order introduced by the sequence of '`&`' operations. Alternatively, a more extensive mesh can be constructed by initially creating a domain `dom` and then applying h-refinement to it using the `refine(dom)` function. Importing meshes from specialised meshing software is supported to a limited extent.

The implementation is designed to accommodate second-order PDOs, specified by coefficients in the format `{{uxx, uxy, uyy}, {ux, uy}, u}` to represent the equation:

$$\texttt{uxx}\frac{\partial^2 u}{\partial x^2} + \texttt{uxy}\frac{\partial^2 u}{\partial x \partial y} + \texttt{uyy}\frac{\partial^2 u}{\partial y^2} + \texttt{ux}\frac{\partial u}{\partial x} + \texttt{uy}\frac{\partial u}{\partial y} + \texttt{u}u = \texttt{rhs}.$$

These coefficients and the righthand side, `rhs`, can be either a scalar or a function handle. To construct an `ultraSEM` object, the domain, PDO, righthand side, and polynomial order are provided.

Upon invoking the `ultraSEM` constructor, local operators on each element are initialised and represented as `ultraSEM.Leaf` objects in the hierarchy. During an upward pass, the hierarchy of merged operators is constructed using the `build` command, creating a tree of `ultraSEM.Parent` objects. If the build stage is not explicitly invoked, it automatically takes place when the user requests a solve operation.

For solving, boundary conditions need to be provided, either as a scalar or a function handle, specifying the boundary conditions on the entire domain, or as an array containing the conditions for each edge separately. The solve stage is executed using the `solve` command (or alternatively, the '\' operator). This computes the solution by applying the hierarchy of operators in a downward pass. The solution is then returned as an `ultraSEM.Sol` object, which offers various functions for plotting, such as `plot` and `contour`, and options for evaluation, like `feval` and `norm`.

Below, we demonstrate the syntax of `ultraSEM` for solving the inhomogeneous Helmholtz equation $\nabla^2 u + 50u = -1$ on a square with zero Dirichlet boundary conditions. The resulting solution is visualised in Figure 4.8. Further examples are provided in Appendix A.

```
% construct domain
v = [-1 1 -1 1];
dom = ultraSEM.rectangle(v);
n = 21;
% PDO
pdo = {{1,0,1},{0,0},50};
% boundary conditions
bc = 0;
% righthand side
rhs = -1;
% solve
S = ultraSEM(dom,pdo,rhs,n);
u = S\bc;
% plot
plot(u)
```

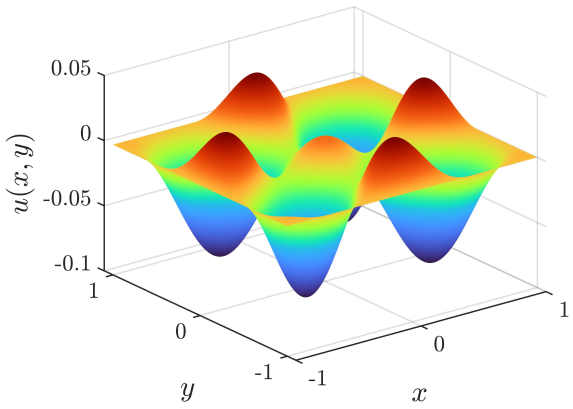

Figure 4.8: The solution to the constant coefficient Helmholtz equation $\nabla^2 u + 50u = -1$ on $[-1, 1]$ subject to zero Dirichlet boundary conditions is depicted. The syntax for calculating this solution using `ultraSEM` is displayed on the left.

CHAPTER 5

NON-RECTANGULAR ELEMENTS

In the preceding chapter, we explored the domain decomposition strategy known as the HPS method, which expanded the capabilities of spectral methods to include a wider range of domains. This approach, as described thus far, only allows for decomposition into rectangular subdomains. In this chapter, we describe how the method can be extended to more complex geometries by applying coordinate transformations. We first investigate an existing transformation for general quadrilaterals in `ultraSEM` and, subsequently, introduce a new geometric shape (the "squonut"), for which we have designed a novel mapping.

Given that spectral methods typically operate on square reference domains, our decision to investigate quadrilaterals over triangles stemmed from the more intuitive mapping it offers. Nevertheless, it is worth noting that mappings from the square to the triangle do exist, with the Duffy transform being one example [60]. Additionally, triangles can be constructed using quadrilaterals through the use of 'kites', as demonstrated in Figure 5.1. Quadrilaterals play a significant role in `ultraSEM` since the default approach for problem-solving on polygons involves their subdivision into quadrilateral elements. Furthermore, when importing meshes for spectral element approximations, the majority of mesh generation programs tend to produce either quadrilateral or triangular elements [45].

5.1 Quadrilateral elements

Typically, utilising spectral methods on irregular geometries involves creating a coordinate transformation that maps points on a reference square to points on a shape in the physical domain [45]. Once the mapping between the physical and reference spaces is established, the next step is to adapt the PDE to accommodate these mappings.

5.1.1 The mapping

Let $\mathcal{Q}$ denote a straight-sided quadrilateral element within the physical space, and let $\mathcal{R} = [-1, 1]^2$ represent the reference square. In the reference space, the coordinates (r, s) is used, while in the real space, the coordinates are denoted by (x, y). Suppose the corners of both shapes are numbered counterclockwise, as shown in Figure 5.1.

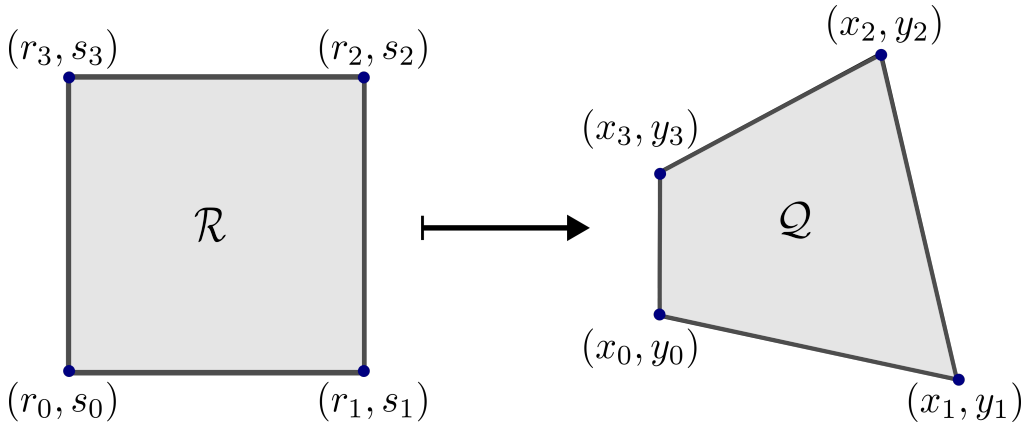

Figure 5.1: A mapping from the reference square $\mathcal{R}$ to a quadrilateral $\mathcal{Q}$ in physical space is depicted. This transformation is described by the bilinear mapping in (5.1) that maps points (r, s) to points (x, y).

To facilitate the mapping from the reference space $(r, s) \in \mathcal{R}$ to the quadrilateral $(x, y) \in \mathcal{Q}$, a bilinear transformation is employed [23], expressed as:

$$\begin{bmatrix} r \\ s \end{bmatrix} \mapsto \begin{bmatrix} a_0^x + a_1^x r + a_2^x s + a_3^x rs \\ a_0^y + a_1^y r + a_2^y s + a_3^y rs \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{5.1}$$

In this transformation, the coefficients $a_0^x, \ldots a_3^x$, and $a_0^y, \ldots a_3^y$ are determined by solving the linear system:

$$\begin{bmatrix} 1 & r_0 & s_0 & r_0 s_0 \\ 1 & r_1 & s_1 & r_1 s_1 \\ 1 & r_2 & s_2 & r_2 s_2 \\ 1 & r_3 & s_3 & r_3 s_3 \end{bmatrix} \begin{bmatrix} a_0^x & a_0^y \\ a_1^x & a_1^y \\ a_2^x & a_2^y \\ a_3^x & a_3^y \end{bmatrix} = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix}. \tag{5.2}$$

It is worth noting that the techniques we employ assume that the boundaries of the quadrilateral domain are smooth, except at the four corners. While it is feasible to create smooth mappings for non-smooth boundaries by solving elliptic systems of PDEs [45], such advanced methods fall outside the scope of our current discussion.

As the transformation from the physical domain to the reference square aligns boundaries with boundaries, applying boundary conditions does not become significantly more complicated. When the boundary constraints are constant values, they remain unchanged by the mapping. However, if a boundary condition is a function, it must be transformed to constrain the solution on the reference square. Likewise, the PDO, which is defined in physical space, must undergo transformation to the reference space where spectral methods can be utilised.

5.1.2 Transforming the equations

To apply a global spectral method on $\mathcal{R}$ to a PDE defined in real space, we need to transform the differential operator $\mathcal{L}$, the righthand side $f(x, y)$, and any nonconstant boundary conditions in the reference space.

The coordinate transformation affects the differential operator via the chain rule. For a function $u(r,s)$ defined on $\mathcal{R}$, the first- and second-order derivatives in x and y are:

$$\begin{aligned} u_x &= r_x u_r + s_x u_s, \\ u_y &= r_y u_r + s_y u_s, \\ u_{xx} &= (r_x)^2 u_{rr} + 2r_x s_x u_{rs} + (s_x)^2 u_{ss} + r_{xx} u_r + s_{xx} u_s, \\ u_{xy} &= r_x r_y u_{rr} + (r_x s_y + r_y s_x) u_{rs} + s_x s_y u_{ss} + r_{xy} u_r + s_{xy} u_s, \\ u_{yy} &= (r_y)^2 u_{rr} + 2r_y s_y u_{rs} + (s_y)^2 u_{ss} + r_{yy} u_r + s_{yy} u_s. \end{aligned} \tag{5.3}$$

The function $f(x,y)$ can be converted into a function $f(r,s)$ by utilising the bilinear transformation in (5.1). When nonconstant Dirichlet boundary conditions are provided, they are transformed in a similar manner as f. However, if Neumann conditions are specified, their derivatives must be transformed using (5.3).

One typically knows the transformation from the reference square to the physical space, but the reverse transformation, i.e., from the physical domain to the reference space, is frequently more complicated and lacks an explicit formula. In most cases, attempting to find the inverse mapping by directly inverting the original mapping function is impractical [23, 45]. As a result, we compute the first-order Jacobian factors $r_x, s_x, r_y,$, and s_y in eq. (5.3) through the use of the inverse function theorem. It shows that the Jacobian matrix $J_{rs} = \partial(r,s)/\partial(x,y)$ can be expressed as $J_{rs} = (J_{xy})^{-1}$, where $J_{xy} = \partial(x,y)/\partial(r,s)$. Explicitly writing out the Jacobians, we derive the following expression for the first-order factors:

$$\begin{bmatrix} r_x & r_y \\ s_x & s_y \end{bmatrix} = \begin{bmatrix} x_r & x_s \\ y_r & y_s \end{bmatrix}^{-1} = \frac{1}{\det(J_{xy})} \begin{bmatrix} y_s & -x_s \\ -y_r & x_r \end{bmatrix}, \tag{5.4}$$

where $\det(J_{xy}) = x_r y_s - x_s y_r$. Utilising the chain rule with these definitions allows one to derive the formulas for the second-order factors $r_{xx}, r_{x,y}, r_{yy}, s_{xx}, s_{xy},$, and s_{yy}. These factors will be rational functions instead of polynomials due to the division by $\det(J_{xy})$, $\det(J_{xy})^2$, and $\det(J_{xy})^3$ during their calculation. Consequently, the coordinate mapping introduces rational variable coefficients into the differential operator. These more complicated coefficients lead to a larger bandwidth when discretising the linear system. Scaling the differential operator $\mathcal{L}$ and the righthand side $f(r,s)$ by the factor $\det(J_{xy})^3$ addresses this aspect and recovers sparsity [23].

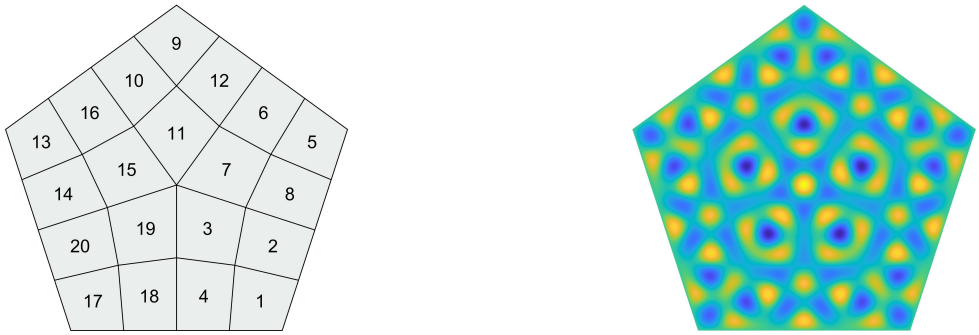

Figure 5.2: On the right, the solution to the constant coefficient Helmholtz equation $\nabla^2 u + 1000u = -1$ with zero Dirichlet boundary conditions on a pentagonal domain is displayed. On the left, the diagram demonstrates the polygonal decomposition into quadrilaterals performed by `ultraSEM` in order to solve.

Having determined the transformed PDO, the global spectral method from Chapter 3 can be employed to solve on $\mathcal{R}$. Finally, the solution is mapped back to the physical domain using the mapping in (5.1), with coefficients $a_0^x, \dots a_3^x$, and $a_0^y, \dots a_3^y$ calculated according to (5.2).

In Figure 5.2, the image on the right presents the solution to the inhomogeneous Helmholtz equation $\nabla^2 u + 1000u = -1$ subject to zero Dirichlet boundary conditions on a pentagonal domain. The syntax for solving this problem in `ultraSEM` is supplied in Appendix A. Regular polygons of this nature are built-in domains within the framework. The approach for solving on these domains involves partitioning them into quadrilateral elements, as demonstrated on the left in Figure 5.2.

5.2 Squonuts

Having considered the extension of the spectral method from the reference square to a general quadrilateral domain, we are prepared to explore more complex geometries. Specifically, we turn our attention to rectangular domains with circular holes, a configuration which is of significant engineering interest. These domains find utility in various scenarios, including the simulation of fluid flow around circular obstacles [45] and the calculation of stress distributions in perforated plates [34]. Additionally, optimisation problems sometimes involve determining the optimal placement of holes, requiring stress and elasticity calculations around such circular boundaries [50]. We focus on a domain consisting of a square with a circular hole in the centre. Due to its unique shape, we refer to it as a square donut. Figure 5.3 sheds some light on the resemblance.

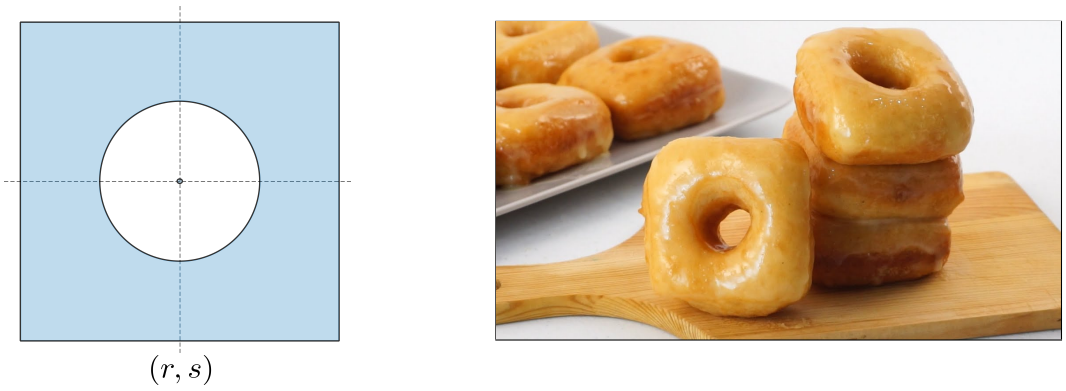

Figure 5.3: The right image depicts a literal square donut. The left illustration shows the domain we refer to when using the phrase "square donut".

We reserve the obvious contraction, "squonut", to refer to a subregion of this domain, which we discuss below. The innovative aspect of this project revolved around the integration of the squonut domain into the `ultraSEM` framework. Achieving this requires the development of an algebraic transformation, akin to the one used for the quadrilateral mapping.

5.2.1 The mapping

To begin, we divided the square donut into smaller, bite-sized pieces. Figure 5.4 shows a specific slice, which is one eighth of the square donut. By replicating, reflecting, and rotating this piece, one can assemble the square donut. Our primary focus was on this specific segment because, with the HPS scheme, one could merge and reconstruct the entire domain. We refer to this slice of a square donut as a "squonut" and denote it by $\mathcal{S}$. Our aim is to establish a mapping from the reference square, $[-1, 1]^2$, to the squonut.

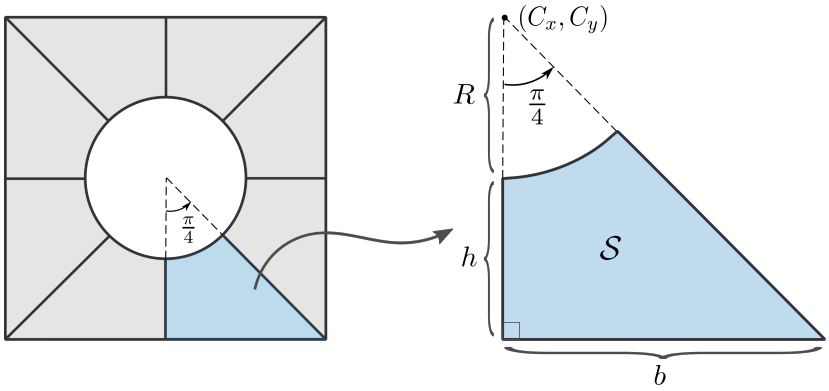

Figure 5.4: A square donut domain is shown on the left. It is partitioned into eight equal-sized parts. One of the slices of the square donut, which is nicknamed a squonut, is shown on the right, together with the parameters that define its size and location.

Our implementation offers the flexibility to create a general squonut, with the only predefined parameter being the angle subtended at the centre of the hole, which is set to $\frac{\pi}{4}$. Users can customise the shape using various input constants. These include the radius of the circular hole, denoted as R, along with its centre position, represented as $\boldsymbol{C} = [C_x, C_y]$, and the height of the left boundary, labelled as h (refer to Figure 5.4). The length of the base is defined as b, which is equal to $h + R$. Moreover, users can specify additional parameters to achieve rotation and reflection of this shape, this is discussed further in Section 5.2.2.

Similar to the approach with quadrilaterals, our mapping ensures that corners and boundaries on the reference square align with those in physical space. A property prioritised in our mapping to the three straight edges of the squonut was the ability to maintain uniform spacing. That is, when points are uniformly distributed along a certain boundary in the reference space, the mapping guarantees that they will also be uniformly spaced along the corresponding boundary in the physical domain. This uniformity is particularly valuable for domain refinement and merging with other domains. Merging the squonut with circular domains at its curved edge is a lower priority, therefore uniformly spaced points on the reference square do not transform into uniformly spaced points on the curved boundary.[1] Figure 5.5c illustrates how uniformly spaced straight lines in the reference square map to corresponding lines and curves in the squonut.

[1]This does not preclude merging the squonut with a circular shape at its curved boundary, but it means that refining such a merged shape must be done carefully.

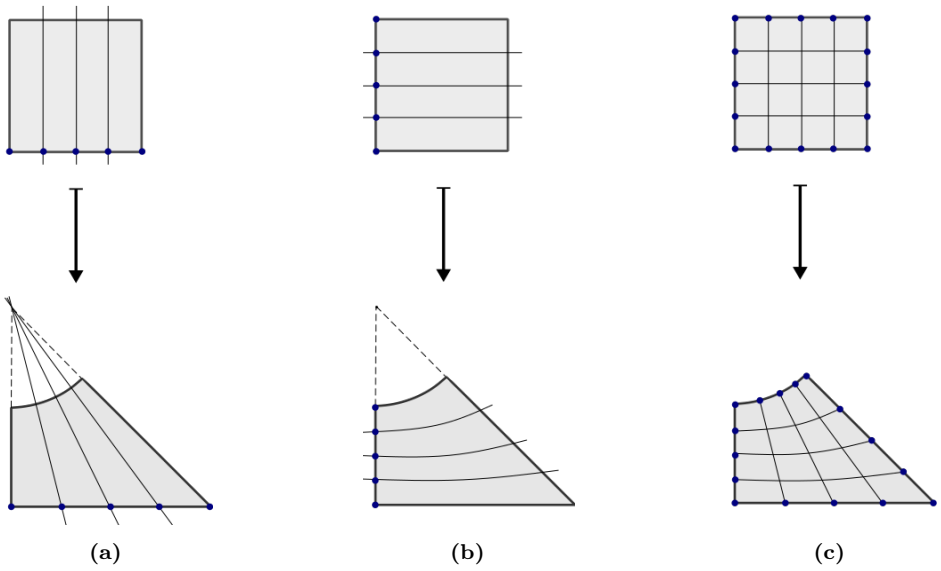

Figure 5.5: Image (a) shows how vertical lines within the reference square map to straight lines traversing through the centre of the squonut. Image (b) illustrates that horizontal lines within the reference space transform into curves within the squonut. The complete mapping process is elucidated in image (c), where one also observes that evenly distributed points on the left, bottom, and right boundaries of the square correspond to evenly spaced points along the three straight edges of the squonut.

To establish a mapping, $\mathcal{M}$, that preserves uniform spacing, the bottom edge of the reference square, which has a side length of 2, is scaled proportionally to match the base of the squonut, which has a length of b. Consequently, a displacement Δr along the bottom edge of the reference square corresponds to a displacement Δx along the bottom edge of the squonut, both measured from the left corner. The relationship between these displacements is $\Delta x = b\frac{\Delta r}{2}$.

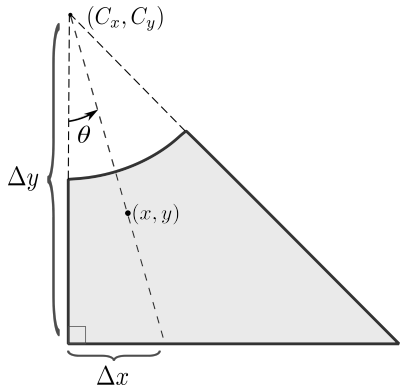

Figure 5.6: A squonut centred at (C_x, C_y) containing a point (x, y). A distance Δx at the base of the squonut subtends an angle θ at the centre.

For the explanation that follows we set $[C_x, C_y] = [0, 0]$, and will expand in Section 5.2.2 on how to translate the shape when a different centre is used. We introduce θ, which is the angle subtended at the origin by the distance Δx. Additionally, Δy denotes the distance on the adjacent side of the resulting triangle, as illustrated in Figure 5.6.

With these definitions in place, we can express:

$$\tan(\theta) = \frac{\Delta x}{\Delta y} = \frac{b\Delta r/2}{b} = \frac{\Delta r}{2} = \frac{r+1}{2}, \tag{5.5}$$

where $\Delta r = r + 1$ since r is a coordinate in the range $[-1, 1]$. Now, for a point (x, y) within the squonut, one can define the relationship as follows:

$$\tan(\theta) = \frac{x}{-y} \quad \Longrightarrow \quad x = -y\tan(\theta) = -y\left(\frac{r+1}{2}\right). \tag{5.6}$$

This equation demonstrates how the x- and y-coordinates are linked in our mapping. The dependence of this relationship on the r-coordinate corresponds to the lines shown in Figure 5.5a. To complete the mapping, we need a relationship that depends on the s-coordinate, as depicted in Figure 5.5b.

The bottom boundary of the squonut forms a straight horizontal line, while the top edge is an arc with radius R centred at the origin. Naturally, a good mapping would incorporate both of these characteristics. We express y as a weighted sum of a straight line, y_ℓ, and a curve, y_c:

$$y = \alpha y_\ell + \beta y_c, \tag{5.7}$$

where the coefficients α and β should ensure that $y(r, -1) = y_\ell$ and $y(r, 1) = y_c$. The horizontal lines y_ℓ have no gradient and depend solely on the s-coordinate. We determine this relationship by scaling the length of the left side of the reference square to match the left edge of the squonut:

$$y_\ell = \frac{h(s+1)}{2} - (R+h) = \frac{h(s-1)}{2} - R, \tag{5.8}$$

where the subtraction of $R + h$ is necessary to adjust for the negative y value measured from the origin. Additionally, one must define y_c, which represents the curves centred at the origin with radii R_s based on the parameter s. These curves align with the points on the left edge of the squonut, akin to y_ℓ. Therefore, the radius of y_c should mirror the magnitude of y_ℓ, just with opposite sign

$$R_s = R + h - \frac{h(s+1)}{2} = R + \frac{h(1-s)}{2}.$$

For points (x_c, y_c) located on these curves, we must have:

$$x_c^2 + y_c^2 = R_s^2.$$

Substituting R_s and using the relationship in (5.6) yields:

$$y_c^2 = \frac{\big(R + h(1-s)/2\big)^2}{1 + (r+1)^2/4}.$$

As y is negative, we assign the negative sign in front of the square root and solve for y_c as:

$$y_c = -\frac{2R + h(1-s)}{\sqrt{4 + (r+1)^2}}. \tag{5.9}$$

To determine the coefficients α and β in (5.7), we enforce that the mapping should transform uniform points on the left and right boundaries of the square into uniformly spaced points on the left and right edges of the squonut. On the left edge, where both y_ℓ and y_c have been chosen to scale uniformly, one must set $\alpha + \beta = 1$ to ensure that y scales uniformly as well. Uniformly spaced points on the right edge of $\mathcal{R}$ should transform to uniformly spaced points on $\mathcal{S}$ given by

$$y = \frac{1}{2}(s+1)\left(R + h - \sqrt{2}R\right) - (R+h),$$

which is determined via trigonometry. Thus, when $r = 1$ then α and β should satisfy

$$y(1,s) = \alpha\left(\frac{h(s-1)}{2} - R\right) - \beta\left(\frac{2R + h(1-s)}{2\sqrt{2}}\right) = \frac{1}{2}(s+1)\left(R + h - \sqrt{2}R\right) - (R+h),$$

which can be solved to find:

$$\beta = \frac{R + Rs}{2R + h - hs}, \quad \alpha = 1 - \beta. \tag{5.10}$$

Having established the mapping, $\mathcal{M}$, from the reference square to the squonut, the derivatives in the PDO can be transformed using the chain rule in (5.3) and the Jacobians in (5.4). Similar to the quadrilateral mapping, the PDO is scaled by the Jacobian determinant $\det(J_{xy})^3$ in order to recover sparser matrices. The global spectral method from Chapter 3 can then be used to solve on $\mathcal{S}$. Finally, the solution is mapped back to the physical domain using the transformation described in (5.7)–(5.10).

5.2.2 Translations, rotations, and reflections

To simplify our derivation, we assumed that the centre of the square donut is located at the origin, however, this does not need to be the case in general. Translating the element from the origin does not impact the calculated α and β coefficients, and, more importantly, it has no influence on the derivatives of the mapping. To incorporate a translation, we shift the existing mapping by the coordinates of the new centre, (C_x, C_y), such that:

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} \mapsto \begin{bmatrix} C_x + C_y\left(\frac{r+1}{2}\right) + x^* \\ C_y + y^* \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}, \tag{5.11}$$

where (x, y) denote the coordinates of the translated squonut and (x^*, y^*) are the coordinates of the squonut centred at the origin.

To build the full square donut domain depicted in Figure 5.4, we must be able to rotate the squonut element. The rotation matrix $R(\phi)$ rotates a set of coordinates counterclockwise by an angle ϕ about its centre. It is defined as

$$R(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}.$$

Applying $R(\phi)$ to the coordinates (x^*, y^*) of a squonut results in a new set of coordinates that represent the squonut rotated counterclockwise by ϕ. Typically, one first rotates the squonut before translating it to a new position, as rotations around the origin are simpler.

Both the rotation and the translation of the squonut after its original mapping can be seen as secondary mappings. The sequence of coordinate transformations can be expressed as:

$$\begin{bmatrix} r \\ s \end{bmatrix} \overset{\mathcal{M}_1}{\longmapsto} \begin{bmatrix} x^* \\ y^* \end{bmatrix} \overset{\mathcal{M}_2}{\longmapsto} \begin{bmatrix} x \\ y \end{bmatrix},$$

where $\mathcal{M}_1$ is the squonut mapping derived in Section 5.2.1 and $\mathcal{M}_2$ the mapping representing rotation or translation (or both).

Another crucial step in constructing the full square donut involves creating a version of the squonut that is mirrored across the y-axis. To achieve this, we consider a copy of the reference square reflected across its left boundary, denoted as $\mathcal{R}_{\text{refl}} = [-1, -3] \times [-1, 1]$. Applying the mapping $\mathcal{M}_1$ to $\mathcal{R}_{\text{refl}}$ results in the reflected squonut, $\mathcal{S}_{\text{refl}}$, as illustrated in Figure 5.7.

Within the `ultraSEM` framework, all transformations are defined from the reference square. Consequently, the sequence of coordinate mappings to the reflected squonut are as follows:

$$\mathcal{R} \overset{\mathcal{M}_0}{\longmapsto} \mathcal{R}_{\text{refl}} \overset{\mathcal{M}_1}{\longmapsto} \mathcal{S}_{\text{refl}}.$$

Here, $\mathcal{M}_0$ can be defined as the operation:

$$\begin{bmatrix} r \\ s \end{bmatrix} \overset{\mathcal{M}_0}{\longmapsto} \begin{bmatrix} -2 + r \\ s \end{bmatrix} = \begin{bmatrix} r_{\text{refl}} \\ s_{\text{refl}} \end{bmatrix}, \tag{5.12}$$

which is responsible for mirroring the reference square along its left boundary.

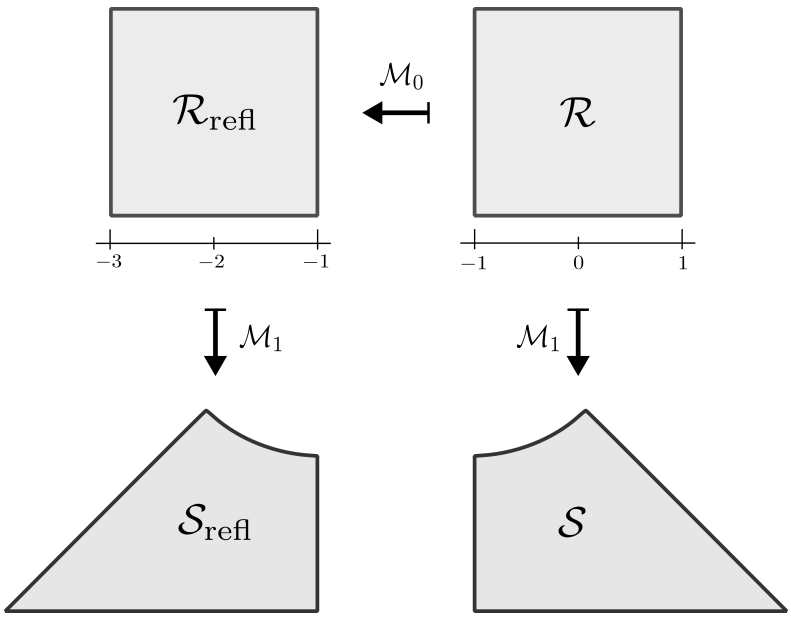

Figure 5.7: The reference square $\mathcal{R}$ is transformed into its reflection using the mapping $\mathcal{M}_0$. Subsequently, the reflected square $\mathcal{R}_{\text{refl}}$ is transformed by the mapping $\mathcal{M}_1$ to produce the reflected squonut $\mathcal{S}_{\text{refl}}$.

5.2.3 Refinement

The squonut, like quadrilateral or triangular domains, can undergo refinement using either p-refinement or h-refinement. When applying p-refinement, the polynomial degree of approximation on the reference square is increased before mapping the solution back to the squonut. Additionally, we employ a uniform h-refinement strategy, which includes dividing the domain in the reference space into four equal partitions and mapping each of these to subdomains of the squonut within the physical space. We refer to squonut subdomains as "subsquonuts". In this h-refinement strategy, the number of sub-elements M in a squonut with refinement level r scales as $M = 2^{2r}$, where $r = 0$ implies no refinement.

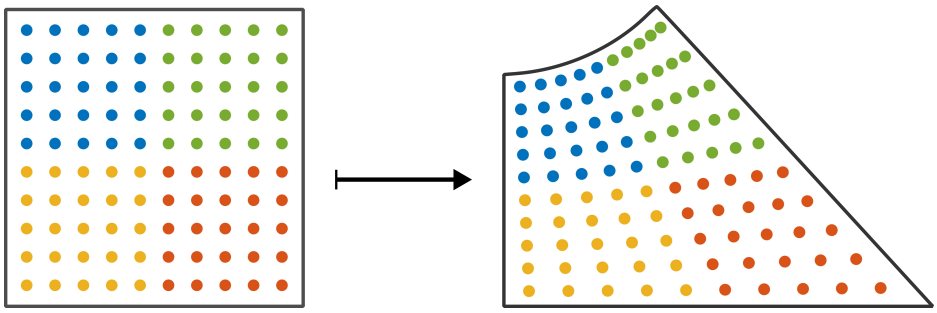

Figure 5.8: A visual representation of the h-refinement strategy employed to refine squonuts. It entails uniformly refining the reference square and mapping the resulting subdomains to corresponding subsquonuts. Note that the points in this illustration serve as a visual aid and do not represent collocation points.

The partitioning for one level of h-refinement can be observed in Figure 5.8. Our choice of this refinement strategy is motivated by two key considerations. Firstly, it simplifies the merging of refined squonuts with other refined shapes along the straight-sided edges. Secondly, given that the mapping from the square to the squonut is known, deducing the mapping of a subdomain from the square to the corresponding subsquonut is straightforward.

However, because all transformations in `ultraSEM` are performed on the reference square, we first establish a mapping, denoted $\mathcal{M}_0$, from the reference square $\mathcal{R}$ to one of its subdomains, represented as $\mathcal{R}_{\text{sub}}$. Afterwards, we apply the standard mapping $\mathcal{M}_1$ to return a subsquonut $\mathcal{S}_{\text{sub}}$:

$$\mathcal{R} \overset{\mathcal{M}_0}{\longmapsto} \mathcal{R}_{\text{sub}} \overset{\mathcal{M}_1}{\longmapsto} \mathcal{S}_{\text{sub}}.$$

If $\mathcal{M}_0$ represents the mapping that returns the left bottom subdomain of the square, it would be given by:

$$\begin{bmatrix} r \\ s \end{bmatrix} \overset{\mathcal{M}_0}{\longmapsto} \frac{1}{2} \begin{bmatrix} r-1 \\ s-1 \end{bmatrix} = \begin{bmatrix} r_{\text{sub}} \\ s_{\text{sub}} \end{bmatrix}. \tag{5.13}$$

Mappings to the other subdomains of the reference square would follow a similar pattern. The derivatives within the subsquonuts remain consistent with those in squonuts, essentially making subsquonuts functionally equivalent to squonuts, and thus they are treated by `ultraSEM` as such.

5.2.4 Software

Our implementation serves as an extension of the existing `ultraSEM` package in MATLAB [23]. The primary objective of this extension is to establish a framework for addressing problems in domains featuring circular holes, by leveraging the efficient and user-friendly structure provided by `ultraSEM`.

The new transformation is encoded as an `ultraSEM.Mapping` object, accessed during the construction of an `ultraSEM.squonut` domain. Nearly all the functionality available to existing domains within the framework has been implemented for the squonut. For instance, the squonut can be merged with copies of itself or with any other domain type using the '&' operator. It supports refinement, either independently or as part of a merged domain, up to a specified refinement level r. Once the squonut domain is constructed, it can be visually represented via the overloaded `plot` function.

To solve on the squonut, a user follows the same process outlined in Section 4.6 for rectangular domains. This involves defining the PDO by specifying its coefficients in the form of a list. This list, along with the domain, righthand side, and polynomial degree, is then passed to the `ultraSEM` constructor. This constructor, in conjunction with the boundary conditions, is then employed to yield the solution in the form of an `ultraSEM.Sol` object. The boundary conditions are used, together with this constructor, to yield the solution in the form of an `ultraSEM.Sol` object, which can also be visualised using the `plot` function. In Figure 5.9, we present a demonstration of the syntax of our implementation to solve the Poisson equation $\nabla u = -1$ on a square donut subject to zero boundary conditions.

```
% construct domain
R = 1; h = 1;
C = [0,0];
S = ultraSEM.squonut(R,h,C);
n = 21;
% PDO
pdo = {{1,0,1},{0,0},0};
% boundary conditions
bc = 0;
% righthand side
rhs = -1;
% solve
A = ultraSEM(S,pdo,rhs,n);
u = A\bc;
% plot
plot(u)
```

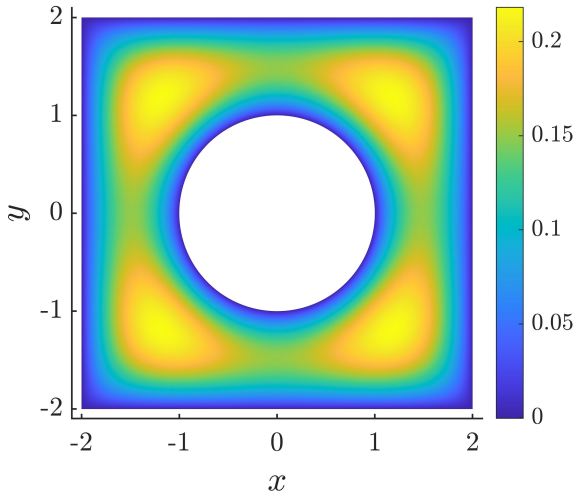

Figure 5.9: The solution to the Poisson problem $\nabla^2 u = -1$ on the square donut is depicted. Zero Dirichlet conditions are enforced on all domain boundaries. The syntax for calculating this solution using the extended version of `ultraSEM` is displayed on the left.

CHAPTER 6

RESULTS

In this chapter, we assess the accuracy and efficiency of our approach before applying it to a range of practical applications. We begin by solving a standard test problem with a known exact solution on the squonut and evaluating the performance. Our analysis includes a comparison of results, such as convergence rates and execution times, with those observed by Fortunato et al. on a rectangular domain [23]. The goal is to evaluate the performance of our method in terms of both accuracy and computational cost, thereby determining its potential as a valuable extension to `ultraSEM`.

While our approach was not originally tailored for fluid dynamics problems, we find practical value in applying it to model fluid flow over and around circular boundaries. Numerous other approaches, including the marker-and-cell method, the vorticity stream function method, and the immersed boundary method, exist specifically for simulating fluid flow over complex immersed bodies [35, 37, 43]. Our approach, on the other hand, is designed for general use, but we demonstrate its efficiency and applicability through these fluid flow problems in various scenarios.

The first practical application we consider involves modelling the transport of contaminant concentration. Following this, we explore solutions arising from a system of reaction-diffusion equations. Subsequently, we investigate the dynamics of the well-known 2D wave equation and apply our method to the 2D Burgers equation. However, before looking into these applications, we first turn our attention to a few numerical experiments.

6.1 Numerical experiments

For our analysis, we compare our approximate solution on the squonut to the exact solution of a Poisson test problem given by:

$$\nabla^2 u = f(x, y), \tag{6.1}$$

where the righthand side is defined as:

$$f(x, y) = -2x^3 - 18x^2y - 28x^2 - 18xy^2 + 2x - 2y^3 - 4y^2 + 2y + 4,$$

subject to zero Dirichlet boundary conditions. This problem is engineered to possess the analytical solution:

$$u_{\text{exact}} = -x(x + y)(y + 2)(x^2 + y^2 - 1).$$

The approximate solution on the squonut is computed with `ultraSEM` and is presented visually on the right in Figure 6.3. As an initial experiment, the execution time is recorded for each stage of the HPS method:[1] initialising operators on leaves, building operators hierarchically on parents, and passing boundary conditions down the hierarchy to solve. In Chapter 4 we discussed the anticipated complexity for these stages when solving within a rectangular domain. For the squonut, one can expect to encounter similar complexity trends, albeit slightly heightened due to the non-trivial derivatives of the coordinate mapping. In Figure 6.1 (left), execution times are displayed for increasing values of n, which is referred to as p-refinement, representing the refinement of the polynomial degree. On the right, the time taken for various levels of uniform domain refinement is presented, known as h-refinement. The number of sub-elements M in a squonut with a refinement level r scales as $\mathcal{O}(2^{2r})$. This can be related to the minimum average element width, h, in terms of which M will scale as $\mathcal{O}(1/h^2)$.

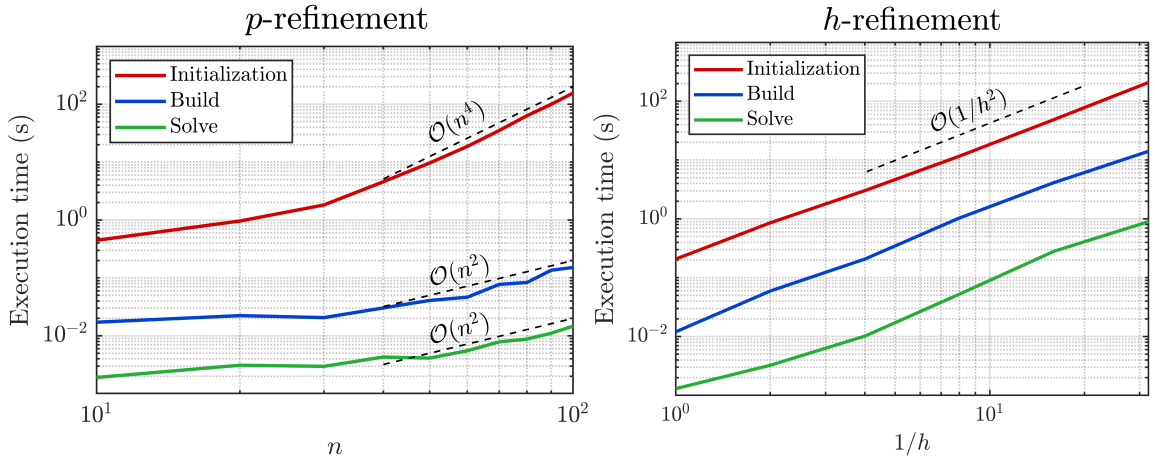

Figure 6.1: The execution time in seconds for the different stages of the HPS scheme when solving Poisson's equation on a squonut. On the left p-refinement is applied, and as predicted based on the analysis of Section 4.4, a rate of increase of $\mathcal{O}(n^2)$ for the build and solve stages is observed, while the initialisation stage has a higher rate of $\mathcal{O}(n^4)$. On the right, h-refinement is applied. The stages all increase at a rate of $\mathcal{O}(1/h^2)$.

In both illustrations, it is evident that the initialisation stage consumes the largest proportion of time. This aligns with our expectations, since a substantial computational workload is required to construct operators on each patch. When p-refinement is applied, the execution time of the initialisation stage increases at a rate of $\mathcal{O}(n^4)$, while for the build and solve stages, the time appears to increase following $\mathcal{O}(n^2)$. However, it is worth remarking again that the initialisation stage is trivially parallelisable. These rates for the squonut are consistent with the complexities calculated for the rectangular element, albeit with a constant scaling factor. When refining the domain, the rate of increase for the different stages adheres to $\mathcal{O}(1/h^2)$, aligning with expectations derived from rectangular elements.

For error calculations, we construct a mesh of 20×20 Chebyshev points on the reference square, subsequently mapping them to points in the physical space. The approximate solution is then compared to the exact solution at these mapped points, and the 2-norm is computed. We conduct error computations for three different refinement levels, $r = 0$, $r = 1$, and $r = 2$, and increasing polynomial degrees. A refinement level of zero indicates no domain refinement,

[1] Experiments were performed on a 1.80GHz Intel Core i7 16GB RAM device with MATLAB R2022b.

while $r = 1$ implies the domain has been partitioned once, resulting in $n_{\text{elem}} = 4$. Our refinement scheme uniformly partitions the reference square, akin to how `ultraSEM` operates for rectangles, causing the number of elements to scale as $\mathcal{O}(1/h^2)$. If N represents the number of degrees of freedom in our approximation, then N scales as $\mathcal{O}(n^2/h^2)$.

The spectral method applied to the squonut demonstrates geometric convergence for p-refinement. This is clearly depicted on the left in Figure 6.2, showing the convergence of three distinct refinement levels. A higher refinement level noticeably accelerates the convergence rate; however, it also comes at the expense of increased computational costs.

In cases without any domain refinement, the error reaches 10^{-13} at a polynomial degree of approximately $n = 20$. Notably, the solution u_{exact} is a bivariate polynomial of degree four in both x and y. Consequently, it can be precisely represented as a bivariate Chebyshev expansion of degree four. Given this, one might anticipate that an approximate solution should converge and provide close to 16 digits of accuracy at around $n = 4$. However, even though a bivariate polynomial approximation is employed on the reference domain, the nonlinear property of the mapping, particularly its non-polynomial nature, implies that the basis used for approximating the solution is not polynomial. Consequently, larger values of n is required for the solution to be accurate. Nonetheless, this does not pose a problem, as solutions in general are not polynomial.

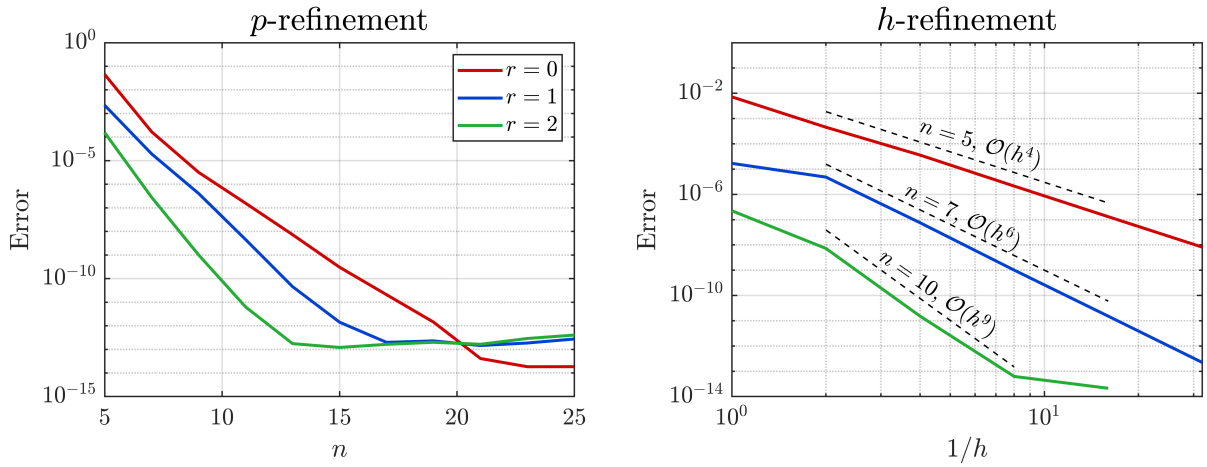

Figure 6.2: On the left is the convergence of `ultraSEM` on the squonut under refinement of n for different h values. The plateau observed for each of the refinement levels is contributed to rounding errors. On the right is the convergence under refinement of h for different n values.

For a constant polynomial degree, algebraic convergence is observed as h is refined. The right image of Figure 6.2 displays the convergence when $n = 5$, $n = 7$, and $n = 10$. The observed error scales roughly as $\mathcal{O}(h^{n-1})$, consistent with the analysis on the rectangular domain [23]. For this example, it appears the estimate holds true for lower values of n but becomes less accurate as n grows.

For this simple problem that possess a smooth solution, the no h-refinement strategy ($r = 0$) manages to capture all essential details of the solution while demanding a lower computational cost than the alternatives, at most p-values at least. This means, that in this case where the solution is not too complicated, p-refinement seems to be a better choice than h-refinement.

However, this might not always be the case for more complicated problems, for example domains with re-entrant corners, where a higher spatial resolution may be needed.

To compare the performance of various combinations of h- and p-refinements, we consider the total flop count of each refinement strategy and the accuracy it achieves. Typically, employing a combination of h- and p-refinement is referred to as a hp-refinement strategy [6, 42]. In most element methods, determining an optimal refinement combination for attaining high accuracy at low computational cost is challenging. Some methods implement adaptive strategies [73], often involving refining specific elements repeatedly instead of the entire domain to enhance accuracy with minimal additional cost.

One can anticipate the total complexity of solving on the squonut to be close to that of rectangular domains, i.e., $Nn^2 + N^{3/2}$. We use this as an estimated number of floating-point operations (called flops) to compare the performance of different refinement combinations. Typically, an hp-adaptivity strategy is determined, that specifies an optimal combination of h- and p-refinement that should be used to reach a certain accuracy. From our results, we observe that a strategy exist for which the error decays super-algebraically in the total flop count C, roughly following $\mathcal{O}(e^{-0.9C^{0.32}})$. This is depicted on the left in Figure 6.3. However, we have not explored further into describing a general strategy in detail.

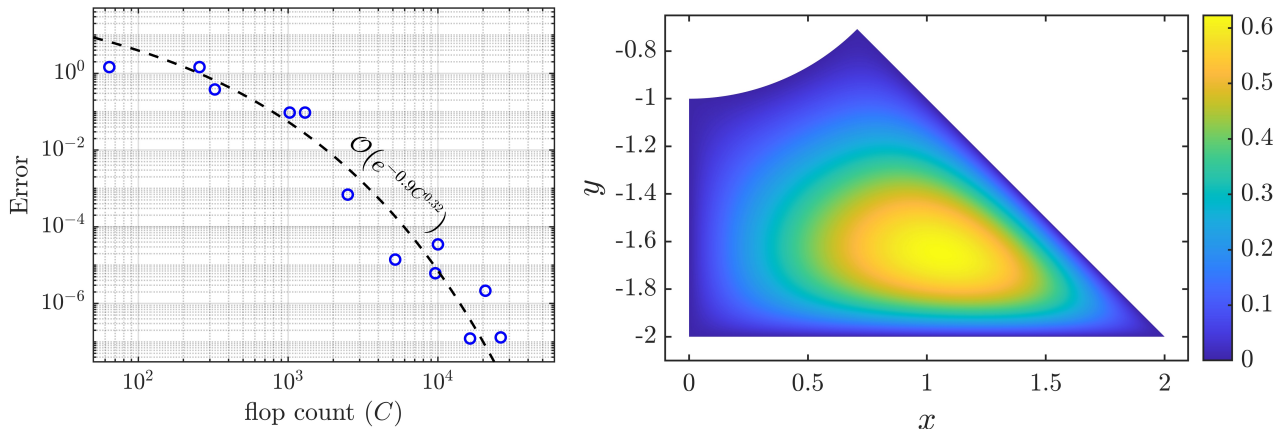

Figure 6.3: On the left, we observe an hp-adaptivity strategy for solving Poisson's equation on the squonut subject to Dirichlet boundary conditions. In this strategy, it appears the error decays super-algebraically in C, which denotes the number of floating-point operations. On the right, the solution for the Poisson problem on the squonut is displayed, computed for $r = 1$ and $n = 15$.

.

6.2 Applications

We now dive into various time-dependent applications within our new domain. In Chapter 3, a backward Euler time-discretisation was employed for linear problems due to its simple and stable nature. However, in the ensuing examples, we encounter nonlinear problems, involving both linear and nonlinear terms, prompting us to consider different discretisation approaches. Implicit-explicit (IMEX) schemes present a suitable strategy, widely utilised by researchers, particularly in conjunction with spectral methods [10, 44]. These schemes prove effective when portions of the equation are stiff and linear, while others are nonlinear and possibly less

stiff. IMEX approaches are favoured over fully implicit discretisations, which face challenges in constructing iterative solvers due to the properties of the matrix to be inverted [2]. In this work, we adopt an IMEX approach that employs a finite difference scheme to approximate the time derivative and discretises linear terms implicitly and nonlinear terms explicitly.

Both backward Euler and IMEX methods necessitate solving one linear system per time step. In the upcoming examples, it becomes evident that the righthand side of the linear system changes at each step, while the operator on the left typically remains constant. In cases like these, `ultraSEM` offers a built-in function called `updateRHS`, designed to facilitate efficient updates. This function enables an object to be updated inexpensively to solve with a new righthand side. Recall that in every solution operator and Dirichlet-to-Neumann operator, the last column is calculated from the particular solution. When employing a new righthand side, a new particular solution must be calculated. This is done in `updateRHS`, which then updates the last column of each of the operators on the leaves and then executes a modified build stage to update the interfacial operators. Figure 6.5 (right) demonstrates the efficiency of the `updateRHS` function when employed to calculate the solution of the contaminant flow problem in Section 6.2.1. From this illustration, it is evident that when dealing with a significant number of time steps, utilising `updateRHS` is advisable.

6.2.1 Transport of a contaminant concentration

As a first problem, we solve a convection-diffusion equation used for modelling the transport of a contaminant concentration within a fluid flow. This model [46] was utilised by Fortunato et al. [23] to show that `ultraSEM` can solve time-dependent problems on simple rectangular domains. We now introduce a cylindrical obstacle to the domain and use the extended software to solve

$$u_t = \kappa \nabla^2 u - \nabla \cdot (\boldsymbol{b}(x,y)u),$$

on the domain Ω over the time span $[0, T]$. The initial condition is set to be

$$u(x,y,0) = e^{-4(x-1)^2-4y^2},$$

along with zero Dirichlet boundary conditions. Additionally, we set the diffusivity to $\kappa = 0.01$ and the convective velocity $\boldsymbol{b}(x,y) = (1 - e^{\gamma x}\cos 2\pi y, \frac{\gamma}{2\pi}\sin 2\pi y)$, where $\gamma = \mathrm{Re}/2 - \sqrt{\mathrm{Re}^2/4 - 4\pi^2}$ and $\mathrm{Re} = 100$ is the Reynolds number. This velocity field $\boldsymbol{b}(x,y)$ corresponds to the analytical solution for the Kovasznay flow on a rectangular domain [46].[2]

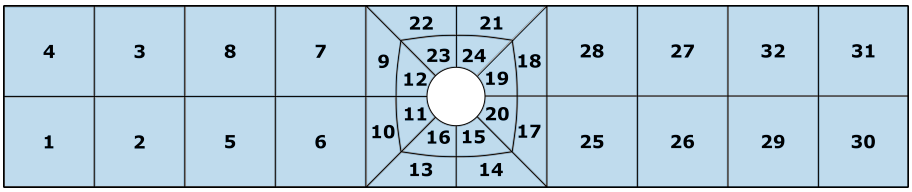

Figure 6.4: The refinement scheme of the domain Ω, with labels indicating the merging order.

[2] Clearly, the domain in this case is not rectangular, hence the model here is not physically accurate. Rather than solve for the velocity field on this domain, we accept this physical inaccuracy so that we might more readily compare the performance of the method with the example from [23].

We denote by $u^{[i]}$ the approximate solution at time $t = i\Delta t$ for integers $i \geq 0$ and a time step Δt. Discretising in time using the backward Euler method yields a steady-state PDE in $u^{[i+1]}$,

$$\left(1 + \Delta t \nabla \cdot \boldsymbol{b} + \Delta t \boldsymbol{b} \cdot \nabla - \Delta t \kappa \nabla^2\right) u^{[i+1]} = u^{[i]}, \tag{6.2}$$

which must be solved at every time step to calculate $u^{[i+1]}$ from $u^{[i]}$. We solve on the domain Ω, which is a rectangle $[0, 10] \times [-1, 1]$ that contains a cylindrical obstacle with radius $R = 0.3$ centred at $(5, 0)$. The domain is refined to produce subdomains as illustrated in Figure 6.4, labelled to indicate their merging order.

Using a polynomial degree of $n = 21$ and a time step $\Delta t = 0.1$, the solution is computed using the extended version of `ultraSEM`. The solution at time points $t = 0$, $t = 3$, and $t = 5$ is displayed on the left in Figure 6.5. Initially, the concentration is confined to the left side of the domain. As time progresses, it gradually disperses, flowing and spreading through the fluid and around the circular obstacle at the centre.

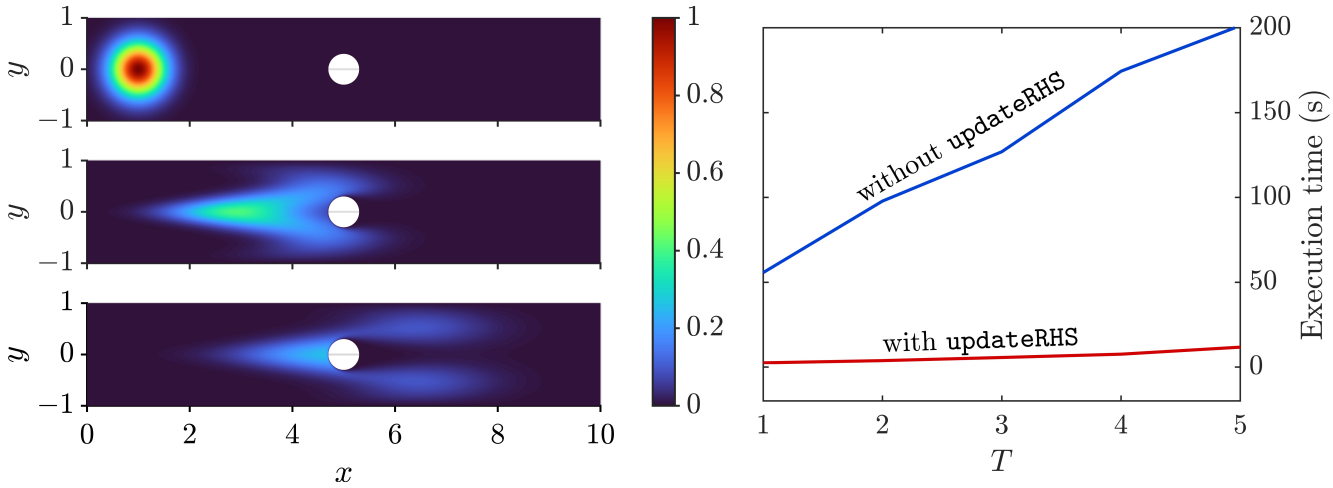

Figure 6.5: On the left the computed solutions at time steps $t = 0$, $t = 3$, and $t = 5$ for the contaminant flow problem in (6.2) are presented. The backward Euler method with a time step of $\Delta t = 0.1$ was employed for time discretisation. The solutions were obtained using the extended version of `ultraSEM` with a polynomial degree of $n = 21$. The domain refinement scheme is depicted in Figure 6.4. On the right, the effect of using `updateRHS` is illustrated.

6.2.2 Gray–Scott equations

Reaction-diffusion equations have been extensively investigated by researchers in biology, chemistry, physics, and computer science to elucidate the formation of patterns in various phenomena such as hair follicle spacing, specific types of coral growth, chemical reactions, zebrafish pigmentation, and more [54]. They are renowned for generating dynamic and captivating patterns and behaviours. The Gray–Scott equations are a set of nonlinear coupled reaction-diffusion equations which are used to describe changes in chemical concentrations over time [67]. They serve as a mathematical model to depict how two chemicals could interact while diffusing through a medium. They are given by

$$\begin{aligned} u_t &= \varepsilon_1 \nabla^2 u + b(1 - u) - uv^2, \\ v_t &= \varepsilon_2 \nabla^2 v - dv + uv^2, \end{aligned}$$

where here we set the diffusion constants to $\varepsilon_1 = 0.002$ and $\varepsilon_2 = 0.0001$, and the constants influencing linear growth or decay as $b = 0.04$ and $d = 0.1$. The coupling between the two equations is provided by the nonlinear term, uv^2, which transfers energy from u to v.

We solve on the square donut domain with a side length of two and a central circular cavity with radius $R = 0.3$. Zero boundary conditions are imposed on all edges, including the curved boundary.

To discretise in time the linear terms in both equations is treated implicitly, and the nonlinear terms explicitly, resulting in the IMEX scheme:

$$\begin{aligned}(1 + \Delta t b - \Delta t \varepsilon_1 \nabla^2) u^{[i+1]} &= u^{[i]} + \Delta t b - \Delta t u^{[i]} \left(v^{[i]}\right)^2, \\ (1 + \Delta t d - \Delta t \varepsilon_2 \nabla^2) v^{[i+1]} &= v^{[i]} + \Delta t u^{[i]} \left(v^{[i]}\right)^2,\end{aligned} \tag{6.3}$$

which uses the solutions from the previous time step, $u^{[i]}$ and $v^{[i]}$, in the nonlinear term to calculate the solution at the following time step, $u^{[i+1]}$ and $v^{[i+1]}$. A spatial discretisation is applied by employing spectral operators with a polynomial degree of $n = 35$. Starting with the initial conditions $v = e^{-80\left((x+0.5)^2+y^2\right)}$ and $u = 1 - v$, the evolution of the solution v is depicted at different times in Figure 6.6.

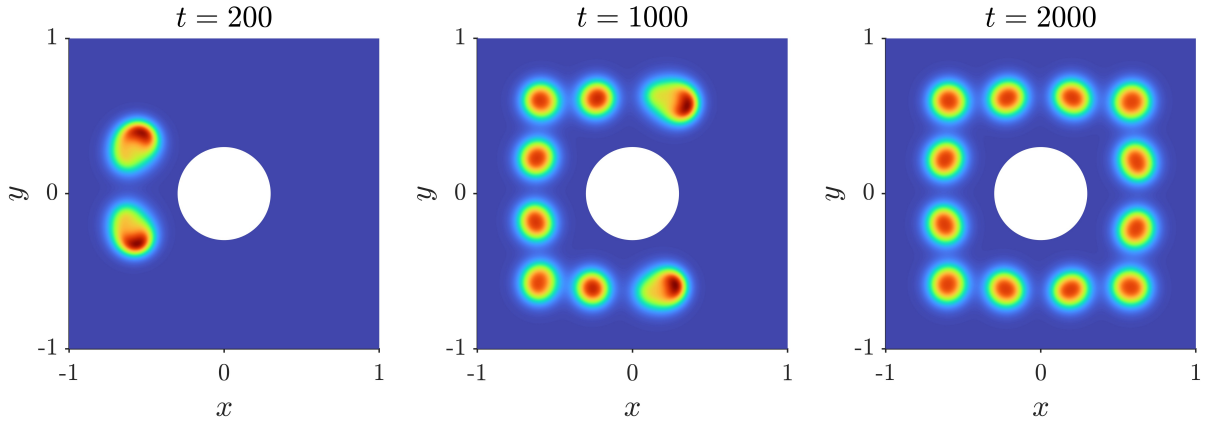

Figure 6.6: The evolution of the v solution in the Gray–Scott equations (6.3) is depicted at time steps $t = 200$, $t = 1000$, and $t = 2000$. An IMEX scheme is employed for the time discretisation with a time step of $\Delta t = 2$. The polynomial degree is $n = 35$ and a refinement level of $h = 1$ is used in `ultraSEM` to solve in space. The initial conditions $v = e^{-80\left((x+0.5)^2+y^2\right)}$ and $u = 1 - v$ were used, along with zero Dirichlet boundary conditions.

6.2.3 The wave equation

The wave equation is a well-known hyperbolic PDE that describes linear, nondispersive wave propagation. It arises in numerous applications, with a classical example being the vibration of an ideal membrane or drum [21]. In 2D, the wave speed is represented by c and the equation takes the form

$$u_{tt} = c^2 \nabla^2 u.$$

This equation is solved on the square donut domain with a side length of two and a circular hole with radius $R = 0.3$ at the centre. We impose zero Neumann conditions on the top, bottom, and curved boundaries, and zero Dirichlet values on the right edge. Along the left edge, we introduce a plane wave via a time-dependent boundary condition: $u(-1, y, t) = e^{-t/\Delta t}$. This models an initial wave entering the domain from the left. Despite the lack of clarity regarding the well-posedness of the wave equation with such boundary data [66], we proceed with it nonetheless.

To discretise in time, a second-order central difference formula is employed and the spatial derivatives is treated implicitly, resulting in the equation:

$$\left(1 - (\Delta t)^2 c^2 \nabla^2\right) u^{[i+1]} = 2u^{[i]} - u^{[i-1]}. \tag{6.4}$$

Here, one must save the solutions from not just one but two previous time steps to calculate the solution at the next step. A time step of size $\Delta t = 0.01$ and a polynomial order of $n = 45$ is used in our computations. The simulation commences with zero displacement and velocity. The solutions at two different times are depicted in Figure 6.7. In this scenario, a wave advances from the left and encounters the circular boundary. It generates a reflected wave that propagates back to the left, forming an arc-shaped pattern, while the main body of the original wave continue to propagate to the right, bypassing the hole.

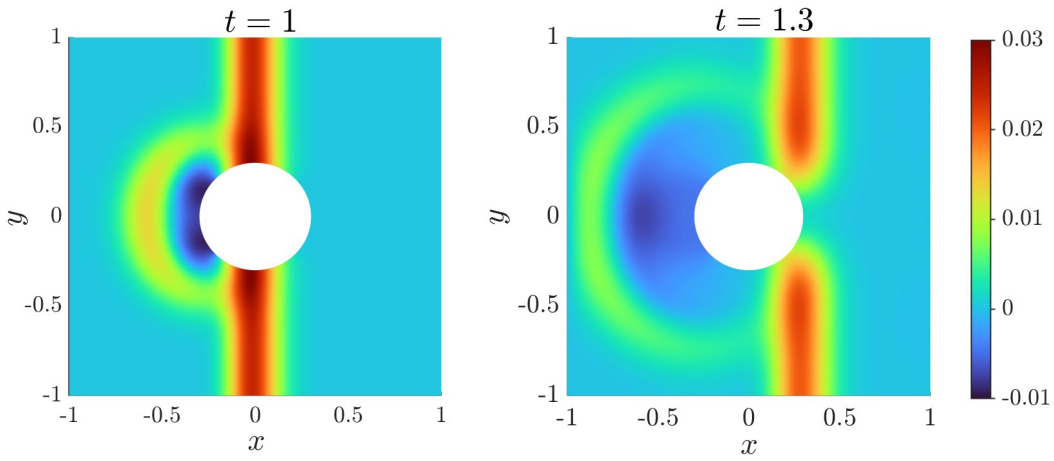

Figure 6.7: The solution to the wave equation (6.4) with $c = 1$ at time steps $t = 1$ and $t = 1.3$ is depicted. A right-moving wave collides with the circular boundary and forms an arc-shaped reflected wave. The solution was computed using $n = 45$ and $\Delta t = 0.01$. A second-order central difference formula was employed for time discretisation. Zero Neumann conditions are imposed on the top, bottom, and curved boundaries, and zero Dirichlet values on the right edge. Along the left edge, a decreasing time-dependent boundary condition, $u(-1, y, t) = e^{-t/\Delta t}$, is enforced. The simulation commences with a zero initial condition.

6.2.4 Burgers' equation

Burgers' equation is a nonlinear PDE of second-order used in various fields to model physical phenomena such as boundary layer behaviour, shock wave formation, turbulence, mass transport, traffic flow, and acoustic transmission [48, 77].

The 2D viscous Burgers' equations are given by:

$$u_t + uu_x + vu_y = \frac{1}{\mathrm{Re}}\nabla^2 u,$$
$$v_t + uv_x + vv_y = \frac{1}{\mathrm{Re}}\nabla^2 v,$$

where here the Reynolds number is taken to be $\mathrm{Re} = 30$. We consider the square $[0, 2]^2$ containing three circular holes, each with radius $R = 0.15$, centred at $(0.5, 1.5)$, $(1.5, 1.5)$ and $(1.5, 0.5)$. This domain is used as a proxy for modelling the flow of shallow water past a collection of cylindrical pillars.

As previously, our time discretisation treats the nonlinear terms explicitly and the linear terms implicitly, giving the system

$$\left(1 - \frac{\Delta t}{Re}\nabla^2\right) u^{[i+1]} = u^{[i]} - \Delta t u^{[i]} u_x^{[i]} - \Delta t v^{[i]} u_y^{[i]},$$
$$\left(1 - \frac{\Delta t}{Re}\nabla^2\right) v^{[i+1]} = v^{[i]} - \Delta t u^{[i]} v_x^{[i]} - \Delta t v^{[i]} v_y^{[i]}.$$

To solve at each time step, the derivatives of the solutions at the previous time step must be calculated. This functionality is built into `ultraSEM` in the form of functions `diffx` and `diffy` that operate on `ultraSEM.Sol` objects. We set $\Delta t = 0.05$, $n = 21$, and enforced zero Neumann boundary conditions. The simulation was initiated using $u = v = e^{-10(0.5-x)^2 - 10(0.5-y)^2}$ and the solution u is depicted at three different times in Figure 6.8.

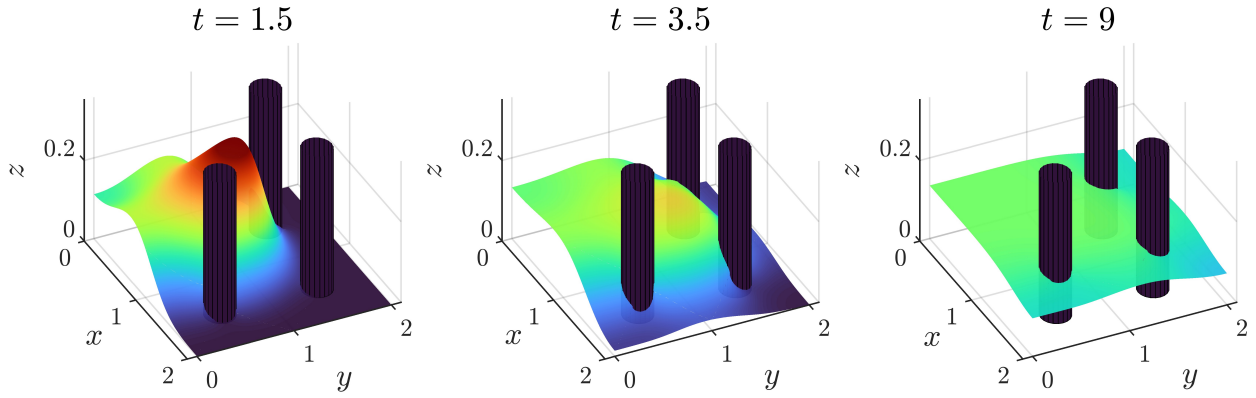

Figure 6.8: The solution to the 2D Burgers equations at time steps $t = 1.5$, $t = 3.5$, and $t = 9$ is depicted. $\Delta t = 0.05$ is used for the time discretisation and $n = 21$ is the polynomial degree for the spatial discretisation.

For reproducibility, the syntax for generating each of the examples in this chapter is provided in Appendix A. These examples highlight the ease and efficiency of the extended software in solving complex time-dependent and fluid dynamics problems, and underscore the practical applicability of the methodology and the new domain.

CHAPTER 7

CONCLUSION

7.1 Summary

This thesis explored spectral methods for the solution of ordinary and partial differential equations. Such equations provide a fundamental framework for modelling diverse phenomena ranging from fluid dynamics and thermodynamics to quantum mechanics and image processing. However, analytical solutions for these equations are often unattainable or impractical due to their complexity, leading to the adoption of numerical techniques.

At the heart of our exploration was the ultraspherical spectral method – a recent advancement in spectral methods. This approach is noted for its sparse linear systems and potential for solving a variety of problems, from linear ordinary differential equations to nonlinear partial differential equations. Yet, like many conventional spectral methods, it is constrained to rectangular domains.

As a first step to overcome this limitation, we investigated the hierarchical Poincaré–Steklov domain decomposition strategy. This approach enhances computational efficiency by segmenting larger domains into manageable components, optimising memory usage, and embracing parallelism, especially in the context of large-scale simulations.

Further extending our exploration, we addressed the constraint of rectangular subdomains by introducing coordinate transformations. In particular, emphasis was placed on the transformation to general quadrilaterals and the introduction of a novel geometric shape – the "squonut". We incorporated this new domain into an existing open-source software package, called `ultraSEM`, which was established for flexible, user-friendly spectral element computations in MATLAB. We demonstrated the feasibility of this extension for solving on rectangular domains featuring circular holes.

To validate our approach, we evaluated its performance and efficiency. Starting with solving a standard test problem, we progressed to practical applications, including modelling contaminant concentration, reaction-diffusion systems, and various systems of differential equations. These applications underscored the practical utility of our extended spectral method.

7.2 Future work

This thesis sought to offer a contribution to spectral methods for solving ordinary and partial differential equations. The presented developments aim to bridge gaps in adaptability to diverse geometries and computational efficiency. The ultraspherical spectral method and the investigated hierarchical Poincaré–Steklov approach have showcased promise in broadening the range of problems amenable to spectral solutions.

However, our work is not exhaustive, and there exist promising avenues for future research. One potential direction is the design of an advanced, non-uniform refinement scheme for the squonut, akin to the scheme implemented by Fortunato et al. [23] for refining into domain corners. Such a scheme would enable the solver to specifically target corners or points where the solution experiences singularities or rapid changes. Achieving this is challenging, as it requires finding a compromise between creating subdomains with known mappings and ensuring that the sides of the squonut will be uniformly partitioned. This refinement scheme would also help facilitate the integration of an adaptive *hp*-refinement strategy, allowing users to specify accuracy levels and thereby enhancing the software's usability.

Exploring additional coordinate transformations for incorporation into the ultraspherical framework offers another possible avenue for advancement. This could potentially broaden the spectrum of domains that can be effectively addressed even further, and enable efficient evaluation on domains whose geometry made it impractical to solve on previously. Shifting the focus towards time-dependent problems and implementing more sophisticated time discretisation schemes, such as higher order IMEX [2] or Runge–Kutta [12] methods, could also expand the application range.

Regarding the `ultraSEM` framework, a natural progression would be extending the methodology to three dimensions, similar to what has been done for collocation methods in the HPS framework [36]. Additionally, investigating the use of impedance-to-impedance mappings [36], especially in scenarios where artificial resonances might be induced during operator merging, has emerged as another feasible extension.

In conclusion, this thesis represents only a fraction of the ongoing evolution and refinement of numerical methodologies, always striving towards the design of accurate, efficient, and versatile numerical solvers.

BIBLIOGRAPHY

[1] N. Alon, J. Bourgain, A. Connes, M. Gromov, and V. Milman, eds., *GAFA 2000*, Birkhäuser Verlag, Basel, 2000. Visions in mathematics. Towards 2000, Geom. Funct. Anal. **2000**, Special Volume, Part I.

[2] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, *Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations*, Appl. Numer. Math., 25 (1997), pp. 151–167.

[3] J. L. Aurentz and R. M. Slevinsky, *On symmetrizing the ultraspherical spectral method for self-adjoint problems*, J. Comput. Phys., 410 (2020), pp. 109383, 24.

[4] T. Babb, A. Gillman, S. Hao, and P.-G. Martinsson, *An accelerated Poisson solver based on multidomain spectral discretization*, BIT, 58 (2018), pp. 851–879.

[5] T. Babb, P.-G. Martinsson, and D. Appelö, *HPS accelerated spectral solvers for time dependent problems: Part I, Algorithms*, in Spectral and high order methods for partial differential equations—ICOSAHOM 2018, vol. 134 of Lect. Notes Comput. Sci. Eng., Springer, Cham, [2020] ©2020, pp. 155–166.

[6] I. Babuška and B. Q. Guo, *Approximation properties of the h-p version of the finite element method*, Comput. Methods Appl. Mech. Engrg., 133 (1996), pp. 319–346.

[7] M. E. Biancolini, *Fast radial basis functions for engineering applications*, Springer, Cham, 2017.

[8] J. P. Boyd, *Chebyshev and Fourier spectral methods*, Dover Publications, Inc., Mineola, NY, second ed., 2001.

[9] J. C. Butcher, *Numerical methods for ordinary differential equations*, John Wiley & Sons, Ltd., Chichester, third ed., 2016. With a foreword by J. M. Sanz-Serna.

[10] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral methods*, Scientific Computation, Springer-Verlag, Berlin, 2006. Fundamentals in single domains.

[11] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral methods*, Scientific Computation, Springer, Berlin, 2007. Evolution to complex geometries and applications to fluid dynamics.

[12] M. H. Carpenter, D. Gottlieb, S. Abarbanel, and W. S. Don, *The theoretical accuracy of Runge-Kutta time discretizations for the initial-boundary value problem: a study of the boundary error*, SIAM J. Sci. Comput., 16 (1995), pp. 1241–1252.

[13] E. W. Cheney, *Introduction to approximation theory*, McGraw-Hill Book Co., New York-Toronto-London, 1966.

[14] L. Cheng and K. Xu, *Solving time-dependent PDEs with the ultraspherical spectral method*, J. Sci. Comput., 96 (2023), pp. Paper No. 70, 34.

[15] C. W. Clenshaw, *A note on the summation of Chebyshev series*, Math. Tables Aids Comput., 9 (1955), pp. 118–120.

[16] B. Cockburn, G. E. Karniadakis, and C.-W. Shu, eds., *Discontinuous Galerkin methods*, vol. 11 of Lecture Notes in Computational Science and Engineering, Springer-Verlag, Berlin, 2000. Theory, computation and applications, Papers from the 1st International Symposium held in Newport, RI, May 24–26, 1999.

[17] J. Crank and P. Nicolson, *A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type*, Proc. Cambridge Philos. Soc., 43 (1947), pp. 50–67.

[18] G. Dhatt and G. Touzot, *The finite element method displayed*, A Wiley-Interscience Publication, John Wiley & Sons, Inc., New York, 1984. Translated from the French by Gilles Cantin.

[19] E. H. Doha and W. M. Abd-Elhameed, *Efficient spectral ultraspherical-dual-Petrov-Galerkin algorithms for the direct solution of* $(2n+1)$*th-order linear differential equations*, Math. Comput. Simulation, 79 (2009), pp. 3221–3242.

[20] J. Fish and T. Belytschko, *A first course in finite elements*, John Wiley & Sons, Ltd., Chichester, 2007. With 1 CD-ROM (Windows).

[21] N. H. Fletcher and T. D. Rossing, *The physics of musical instruments*, Springer-Verlag, New York, second ed., 1998.

[22] D. Fortunato, N. Hale, and A. Townsend, *Github repository*, 2021. https://github.com/danfortunato/ultraSEM.

[23] D. Fortunato, N. Hale, and A. Townsend, *The ultraspherical spectral element method*, J. Comput. Phys., 436 (2021), pp. Paper No. 110087, 20.

[24] D. Fortunato and A. Townsend, *Fast Poisson solvers for spectral methods*, IMA J. Numer. Anal., 40 (2020), pp. 1994–2018.

[25] J. D. Gardiner, A. J. Laub, J. J. Amato, and C. B. Moler, *Solution of the Sylvester matrix equation* $AXB^\top + CXD^\top = E$, ACM Trans. Math. Software, 18 (1992), pp. 223–231.

[26] P. Geldermans and A. Gillman, *An adaptive high order direct solution technique for elliptic boundary value problems*, SIAM J. Sci. Comput., 41 (2019), pp. A292–A315.

[27] M. Geshi, ed., *The art of high performance computing for computational science. Vol. 1—techniques of speedup and parallelization for general purposes*, Springer, Singapore, [2019] ©2019.

[28] A. Gillman and P. G. Martinsson, *A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method*, SIAM J. Sci. Comput., 36 (2014), pp. A2023–A2046.

[29] A. Gillman and P.-G. Martinsson, *An $O(N)$ algorithm for constructing the solution operator to 2D elliptic boundary value problems in the absence of body loads*, Adv. Comput. Math., 40 (2014), pp. 773–796.

[30] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, fourth ed., 2013.

[31] D. Gottlieb and S. A. Orszag, *Numerical analysis of spectral methods: theory and applications*, vol. No. 26 of CBMS-NSF Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[32] L. Greengard, *Spectral integration and two-point boundary value problems*, SIAM J. Numer. Anal., 28 (1991), pp. 1071–1080.

[33] B. Y. Guo, H. P. Ma, W. M. Cao, and H. Huang, *The Fourier-Chebyshev spectral method for solving two-dimensional unsteady vorticity equations*, J. Comput. Phys., 101 (1992), pp. 207–217.

[34] M. F. Gur'ev, *Stress distribution in a stretched isotropic finite rectangular plate weakened by a circular opening*, Dopovidi Akad. Nauk Ukrain. RSR, 1953 (1953), pp. 133–139.

[35] K. E. Gustafson and J. A. Sethian, eds., *Vortex methods and vortex motion*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1991.

[36] S. Hao and P.-G. Martinsson, *A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré-Steklov operators*, J. Comput. Appl. Math., 308 (2016), pp. 419–434.

[37] F. H. Harlow and J. E. Welch, *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, Phys. Fluids, 8 (1965), pp. 2182–2189.

[38] J. S. Hesthaven, S. Gottlieb, and D. Gottlieb, *Spectral methods for time-dependent problems*, vol. 21 of Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, Cambridge, 2007.

[39] U. Hetmaniuk, *Introduction to finite and spectral element methods using MATLAB. Second edition [book review MR3185715]*, SIAM Rev., 57 (2015), p. 161.

[40] K. D. Housiadas and C. Tsangaris, *High-order lubrication theory in channels and tubes with variable geometry*, Acta Mech., 233 (2022), pp. 4063–4081.

[41] T. J. R. Hughes, *The finite element method*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1987. Linear static and dynamic finite element analysis, With the collaboration of Robert M. Ferencz and Arthur M. Raefsky.

[42] G. E. Karniadakis and S. J. Sherwin, *Spectral/hp element methods for computational fluid dynamics*, Numerical Mathematics and Scientific Computation, Oxford University Press, New York, second ed., 2005.

[43] J. Kim, D. Kim, and H. Choi, *An immersed-boundary finite-volume method for simulations of flow in complex geometries*, J. Comput. Phys., 171 (2001), pp. 132–150.

[44] J. Kim and P. Moin, *Application of a fractional-step method to incompressible Navier-Stokes equations*, J. Comput. Phys., 59 (1985), pp. 308–323.

[45] D. A. Kopriva, *Implementing spectral methods for partial differential equations*, Scientific Computation, Springer, Berlin, 2009. Algorithms for scientists and engineers.

[46] L. I. G. Kovasznay, *Laminar flow behind two-dimensional grid*, Proc. Cambridge Philos. Soc., 44 (1948), pp. 58–62.

[47] Z. Li, Z. Qiao, and T. Tang, *Numerical solution of differential equations*, Cambridge University Press, Cambridge, 2018. Introduction to finite difference and finite element methods.

[48] J. D. Logan, *An introduction to nonlinear partial differential equations*, Pure and Applied Mathematics (Hoboken), Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, second ed., 2008.

[49] J. Loustau, *Numerical differential equations*, World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2016. Theory and technique, ODE methods, finite differences, finite elements and collocation.

[50] M. Mair and B. I. Wohlmuth, *A domain decomposition method for domains with holes using a complementary decomposition*, Comput. Methods Appl. Mech. Engrg., 193 (2004), pp. 4961–4978.

[51] P.-G. Martinsson, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 38 (2009), pp. 316–330.

[52] P. G. Martinsson, *A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method*, J. Comput. Phys., 242 (2013), pp. 460–479.

[53] J. C. Mason and D. C. Handscomb, *Chebyshev polynomials*, Chapman & Hall/CRC, Boca Raton, FL, 2003.

[54] A. Mesbahi and S. Mesbahi, *On the existence of periodic solutions of a degenerate parabolic reaction-diffusion model*, Nonlinear Dyn. Syst. Theory, 22 (2022), pp. 197–205.

[55] S. a. Milewski, *Meshless finite difference method with higher order approximation—applications in mechanics*, Arch. Comput. Methods Eng., 19 (2012), pp. 1–49.

[56] M. Mitra and S. Gopalakrishnan, *Wavelet based 2-D spectral finite element formulation for wave propagation analysis in isotropic plates*, CMES Comput. Model. Eng. Sci., 15 (2006), pp. 49–67.

[57] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, and C. W. Clark, eds., *NIST handbook of mathematical functions*, U.S. Department of Commerce, National Institute of Standards and Technology, Washington, DC; Cambridge University Press, Cambridge, 2010. With 1 CD-ROM (Windows, Macintosh and UNIX).

[58] S. Olver and A. Townsend, *A fast and well-conditioned spectral method*, SIAM Rev., 55 (2013), pp. 462–489.

[59] S. Olver, A. Townsend, and G. Vasil, *A sparse spectral method on triangles*, SIAM J. Sci. Comput., 41 (2019), pp. A3728–A3756.

[60] S. A. Orszag, *Spectral methods for problems in complex geometries*, J. Comput. Phys., 37 (1980), pp. 70–92.

[61] C.-Y. Pang, R.-G. Zhou, B.-Q. Hu, W. Hu, and A. El-Rafei, *Signal and image compression using quantum discrete cosine transform*, Inform. Sci., 473 (2019), pp. 121–141.

[62] R. Peyret, *Spectral methods for incompressible viscous flow*, vol. 148 of Applied Mathematical Sciences, Springer-Verlag, New York, 2002.

[63] R. B. Platte and L. N. Trefethen, *Chebfun: a new kind of numerical computing*, in Progress in industrial mathematics at ECMI 2008, vol. 15 of Math. Ind., Springer, Heidelberg, 2010, pp. 69–87.

[64] D. Potts, *Fast algorithms for discrete polynomial transforms on arbitrary grids*, vol. 366, 2003, pp. 353–370. Special issue on structured matrices: analysis, algorithms and applications (Cortona, 2000).

[65] E. Previato, ed., *Dictionary of applied math for engineers and scientists*, Comprehensive Dictionary of Mathematics, CRC Press, Boca Raton, FL, 2003.

[66] M. A. Sadybekov and N. A. Yessirkegenov, *Boundary-value problems for wave equations with data on the whole boundary*, Electron. J. Differential Equations, (2016), pp. Paper No. 281, 9.

[67] S. K. Scott, *Chemical chaos*, vol. 24 of International Series of Monographs on Chemistry, The Clarendon Press, Oxford University Press, New York, 1991.

[68] J. Shen, *Efficient spectral-Galerkin method. II. Direct solvers of second- and fourth-order equations using Chebyshev polynomials*, SIAM J. Sci. Comput., 16 (1995), pp. 74–87.

[69] J. Shen, T. Tang, and L.-L. Wang, *Spectral methods*, vol. 41 of Springer Series in Computational Mathematics, Springer, Heidelberg, 2011. Algorithms, analysis and applications.

[70] J. Shen and L.-L. Wang, *Legendre and Chebyshev dual-Petrov-Galerkin methods for hyperbolic equations*, Comput. Methods Appl. Mech. Engrg., 196 (2007), pp. 3785–3797.

[71] J. Shen and L.-L. Wang, *Some recent advances on spectral methods for unbounded domains*, Commun. Comput. Phys., 5 (2009), pp. 195–241.

[72] M. Tokman, *A new class of exponential propagation iterative methods of Runge-Kutta type (EPIRK)*, J. Comput. Phys., 230 (2011), pp. 8762–8778.

[73] A. Townsend and S. Olver, *The automatic solution of partial differential equations using a global spectral method*, J. Comput. Phys., 299 (2015), pp. 106–123.

[74] A. Townsend and L. N. Trefethen, *An extension of Chebfun to two dimensions*, SIAM J. Sci. Comput., 35 (2013), pp. C495–C518.

[75] L. N. Trefethen, *Spectral methods in MATLAB*, vol. 10 of Software, Environments, and Tools, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[76] L. N. Trefethen, *Approximation theory and approximation practice*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2013.

[77] A.-M. Wazwaz, *Analytic study on Burgers, Fisher, Huxley equations and combined forms of these equations*, Appl. Math. Comput., 195 (2008), pp. 754–761.

[78] F. Zhang, ed., *The Schur complement and its applications*, vol. 4 of Numerical Methods and Algorithms, Springer-Verlag, New York, 2005.

APPENDIX A

CODE EXTRACTS

Helmholtz equation on pentagon

```
% Spatial discretisation:
dom = ultraSEM.polygon(5);
dom = refine(dom);
n = 20;

% PDO:
pdo = {{1,0,1}, {0,0}, 1000};

% RHS:
rhs = -1;

% Boundary conditions:
bc = 0;

% Initialise:
S = ultraSEM(dom, pdo, rhs, n);

% Solve:
u = S \ bc;

% Plot:
plot(u)
```

Transport of a contaminant concentration

```
% Time discretisation:
dt = 0.1; T = 10;
nsteps = ceil(T / dt);

% Spatial discretisation:
sq = ultraSEM.squonut(0.3,0.7,[5,0]);
rect1 = ultraSEM.rectangle([0 4 -1 1]);
```

```
rect2 = ultraSEM.rectangle([6 10 -1 1]);
dom = rect1 & sq & rect2;
dom = refine(dom);
n = 21;

% PDE parameters:
kappa = 0.01; % Diffusivity
Re = 100;     % Reynolds number

% Kovasznay flow velocity field:
gamma = Re / 2-sqrt(Re^2 / 4+4*pi^2);
bx = @(x,y) 1-exp(gamma*x).*cos(2*pi*y);
by = @(x,y) gamma / (2*pi)*exp(gamma*x).*sin(2*pi*y);

% Compute the divergence of the velocity field:
x = chebfun2(@(x,y) x, rect);
y = chebfun2(@(x,y) y, rect);
divb = diffx(bx(x,y)) + diffy(by(x,y));
db1dx = @(x,y) -gamma.*exp(gamma.*x).*cos(2*pi.*y);
db2dy  = @(x,y) gamma.*exp(gamma.*x).*cos(2*pi.*y);

% Backwards Euler:
pdo = {-dt*kappa, {@(x,y) dt*bx(x,y), @(x,y) dt*by(x,y)},
       @(x,y) 1+dt*divb(x,y)};

% Initial and boundary conditions:
u0 = @(x,y) exp(-4*((x-1).^2+y.^2));
bc = @(x,y,t) 0*x;

% Initialise:
u = ultraSEM.Sol(u0, n, dom);
S = ultraSEM(dom, pdo, 0, n);

% Run a simulation:
t = 0;
for i = 1:nsteps
    % update RHS
    S.rhs = u;

    % solve
    u = S \ bc;

    % update time step
    t = t+dt;
end

% Plot the final solution:
```

```
plot(u)
```

Gray-Scott equations

```
% Time discretisation:
dt = 2; T = 200;
nsteps = ceil(T / dt);

% Spatial disretisation:
R = 0.3; h = 0.7;
sq = ultraSEM.squonut(R,h,[0,0]);
dom = refine(sq);
n = 35;

% Initial conditions:
u0 = @(x,y) 1-exp(-80*((x+.5).^2+(y).^2));
v0 = @(x,y) exp(-80*((x+.5).^2+(y).^2));

% PDE parameters:
b = 0.04; d = 0.1;
ep1 = 0.002; ep2 = 0.0001;

% IMEX scheme:
pdou = ultraSEM.PDO({-ep1*dt, 0 ,-ep1*dt}, {0,0}, 1+dt*b);
pdov = ultraSEM.PDO({-ep2*dt, 0 ,-ep2*dt}, {0,0}, 1+dt*d);

% Initial and boundary conditions:
u = u0; v = v0;
bcu = 0; bcv = 0;

% Initialise:
Su = ultraSEM(dom, pdou, 0, n);
Sv = ultraSEM(dom, pdov, 0, n);

% Run a simulation:
t = 0;
for i = 1:nsteps
    % update RHS
    Su.rhs = @(x,y) u(x,y) - h.*u(x,y).*v(x,y).^2 + dt*b;
    Sv.rhs = @(x,y) v(x,y) + h.*u(x,y).*v(x,y).^2;

    % solve
    u = Su \ bcu;
    v = Sv \ bcv;
```

```
    % update time step
    t = t+dt;
end

% Plot the final solution:
plot(v)
```

The wave equation

```
% Time discritisation:
dt = 0.01; T = 2;
nsteps = ceil(T / dt);

% Spatial discretisation:
h = 0.7; R = 0.3;
sq = ultraSEM.squonut(R,h,[0,0]);
dom = refine(sq);
n = 45;

% Backwards Euler:
pdo = {-dt^2, 0, 1};

% Initial condition:
u0 = @(x,y) 0;

% Initialise:
u = ultraSEM.Sol(u0, p, dom);
S = ultraSEM(dom, pdo, 0, n);

% Boundary conditions - manually specify:
bc = BCGUI(L);

% Run a simulation:
t = 0;
for i = 1:nsteps
    % update time-dependant bc
    for idx = [7]
        bc(idx).val = exp(-t / dt);
    end

    % solve
    unew = S \ bc;

    % update previous solution
    uprev = u; u = unew;
```

```
    % update RHS
    S.rhs = (2*u-uprev);

    % update time step
    t = t+dt;
end

% Plot the final solution:
plot(u)
```

Burgers' equations

```
% Time discretisation:
dt = 0.05; T = 9;
nsteps = ceil(T / dt);

% Spacial discretisation:
R = 0.15; h = 0.35;
sq1 = ultraSEM.squonut(R,h,[1.5,0.5]);
sq2 = ultraSEM.squonut(R,h,[1.5,1.5]);
sq3 = ultraSEM.squonut(R,h,[0.5,1.5]);
rect = ultraSEM.rectangle([0 1 0 1]);
dom = sq1 & sq2 & sq3 & rect;
n = 21;

% PDE parameters:
Re = 30;    % Reynolds number

% IMEX scheme:
pdou = ultraSEM.PDO({-dt / Re,0,-dt / Re}, {0,0}, 1);
pdov = ultraSEM.PDO({-dt / Re,0,-dt / Re}, {0,0}, 1);

% Initial conditions:
u0 = @(x,y) exp(-10*((0.5-x).^2 + (0.5-y).^2));
v0 = @(x,y) exp(-10*((0.5-x).^2 + (0.5-y).^2));

% Zero Neumann boundary conditions:
bcu = ultraSEM.BC(0,0,1);
bcv = ultraSEM.BC(0,0,1);

% Initialise:
u = ultraSEM.Sol(u0, n, dom);
v = ultraSEM.Sol(v0, n, dom);
Su = ultraSEM(dom, pdou, 0, n);
```

```
Sv = ultraSEM(dom, pdov, 0, n);

% Run a simulation:
t = 0;
for i = 1:nsteps
    % update derivatives
    dudx = diffx(u); dudy = diffy(u);
    dvdx = diffx(v); dvdy = diffy(v);

    % update RHS
    Su.rhs = u.*(1-dt.*dudx) - dt.*v.*dudy;
    Sv.rhs = v.*(1-dt.*dvdy) - dt.*u.*dvdx;

    % solve
    u = Su \ bcu;
    v = Sv \ bcv;

    % update time step
    t = t+dt;
end

% Plot the final solution:
plot(u)
```