Accelerated time-stepping of parabolic and hyperbolic PDEs via fast direct solvers for elliptic problems

by

T. Babb

B.S., University of Colorado, 2013

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Applied Mathematics
2019

This thesis entitled:
Accelerated time-stepping of parabolic and hyperbolic PDEs via fast direct solvers for elliptic problems
written by T. Babb
has been approved for the Department of Applied Mathematics

Prof. Per-Gunnar Martinsson

Prof. Daniel Appelö

Prof. Adrianna Gillman

Prof. Bengt Fornberg

Prof. Gregory Beylkin

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Babb, T. (Ph.D., Applied Mathematics)

Accelerated time-stepping of parabolic and hyperbolic PDEs via fast direct solvers for elliptic problems

Thesis directed by Prof. Per-Gunnar Martinsson

The dissertation concerns numerical methods for approximately solving certain linear partial differential equations. The foundation is a solution methodology for linear elliptic boundary value problems that we call the "Hierarchical Poincaré-Steklov (HPS)" method. This method is based on a high-order multidomain spectral discretization that is designed to work particularly well in conjunction with nested-dissection type direct solvers. The methods presented apply in any dimension, but their efficiency deteriorates as the dimension increases, and dimensions higher than three are generally not considered.

A key competitive advantage of the HPS method is that the linear system that results from discretizing an elliptic PDE is solved using a direct rather than an iterative solver. This solver is closely related to existing nested dissection and multifrontal solvers, and has a similar computational profile that involves a "build stage" that is reasonably efficient, and then a "solve stage" that is *very* fast. This makes the method particularly powerful for use in situations where a sequence of linear problems involving the same operator needs to be solved, as happens for instance when solving certain parabolic and hyperbolic PDEs. The use of a direct solver also enables the method to solve many problems that are intractable to iterative solvers, such as Helmholtz problems at intermediate and high frequencies.

The HPS methodology was originally published as a solution method for homogeneous elliptic problems, and the core contributions of the dissertation involve the extension of the methodology to more general environments. Specifically, there are four key contributions:

(1) An extension of the method to handle non homogeneous elliptic equations that involve forcing terms in the volume of the domain.

(2) A generalization of the method to allow the use of refined meshes in order to resolve local singularities.

(3) An efficient solver for hyperbolic equations that works by applying the HPS methodology to explicitly build highly accurate approximations to the time evolution operator. This enables the use of very long time steps, and parallel in time implementations.

(4) An efficient solver for parabolic problems, where the main idea is to accelerate implicit time-stepping schemes by using the HPS methodology to pre-compute the solution operator involved in the elliptic solve. This work also includes an extension to certain non-linear problems.

All techniques presented are analyzed in terms of their complexity. Accuracy and stability are demonstrated via extensive numerical examples.

Dedication

To the woman who brought me joy and happiness in the final months of this dissertation. The kindness you have shown to me means more than you will ever know. You have always seen me and treated me in a way that nobody else has. You make me feel better about myself. In the final weeks of this dissertation I absolutely did not think I could finish this, but it was your smile and kindness that motivated me to push through to the end. Thank you for being my friend.

Acknowledgements

Thank you to the people in GRU who gave me a home and one place where I was actually happy. In particular thank you to Emily, Bekah, Sam, Josh, Ali, Michele, Leonard, Lucy, Teacup, Ola, Luda, Julia, Tristan, Christian, and James. Many of you will never know the impact that your kindness had on my life, but thank you for making me feel accepted.

Thank you Tyler, Katie, Matt, and everyone in group for being some of my best friends when I needed it the most. I wouldn't have made it through the hardest 18 months of my life without all of you.

Thank you to everyone in the department who gave me your time during my ten and a half years here. One day I received an email from Harvey Segur saying "I think that you have every right to feel quite proud of what you accomplished." The professors show an outstanding level of support and genuine care for the students. Thank you Anne Dougherty for everything.

Thank you Gunnar for all the years of work and support you gave me. I would still be directionless and not even know where to start without you.

Thank you Daniel Appelö for doing way more than what was required to save this dissertation from the dumpster. I would still be sitting in my office with my head in my hands unable to finish if it wasn't for you. Thank you for listening when my life wasn't happy.

Thank you Rod Kuseski for being my friend these past few years. You understand me better than anyone I've ever known and you are strangely one of the best friends I've ever had.

Thank you Mom and Dad for all your love and support. I never could have been so ambitious and pursued so many things without your support.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

1.1 Direct solvers for elliptic PDEs

The dissertation describes a set of recently developed *direct* solvers for linear elliptic PDEs, and the application of such solvers to certain time-dependent partial differential equations. In this section, we will briefly introduce some of the key ideas underlying direct solvers, and discuss how they offer exciting new capabilities that complement existing iterative solvers. To frame the discussion, let us consider a simple model boundary value problem

$$\begin{cases} [Au](\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma, \end{cases} \tag{1.1}$$

where Ω is a domain in $\mathbb{R}^2$ with boundary $\Gamma = \partial\Omega$, and where A is an elliptic operator of the form

$$\begin{aligned} [Au](\boldsymbol{x}) = -c_{11}(\boldsymbol{x})[\partial_1^2 u](\boldsymbol{x}) - 2c_{12}(\boldsymbol{x})[\partial_1\partial_2 u](\boldsymbol{x}) - c_{22}(\boldsymbol{x})[\partial_2^2 u](\boldsymbol{x}) \\ + c_1(\boldsymbol{x})[\partial_1 u](\boldsymbol{x}) + c_2(\boldsymbol{x})[\partial_2 u](\boldsymbol{x}) + c(\boldsymbol{x})\, u(\boldsymbol{x}). \end{aligned} \tag{1.2}$$

We say that a numerical method for solving (1.1) is *fast* if its algorithmic complexity scales as $\mathcal{O}(N \log^k N)$, where N is the number of degrees of freedom in the discretization, and where k is a small integer, usually $k = 0, 1, 2$. As the available random access memory in a standard computer has increased over the years the problems that we can solve have become much more complex. We can now handle problems with orders of magnitude more degrees of freedom than what was possible 30 years ago, but this means the algorithmic complexity becomes more important. This is why we seek fast solvers for PDEs.

We say that a numerical method for solving (1.1) is *direct* if it in a single sweep constructs an approximation to the solution operator of (1.1). In practical terms, let

$$\mathsf{\mathbf{A}}\mathsf{\mathbf{u}} = \mathsf{\mathbf{b}}, \tag{1.3}$$

denote a linear system arising upon discretization of (1.1). A direct solver takes a computational tolerance ϵ as an input parameter and builds a matrix $\mathsf{\mathbf{S}}$ such that

$$\|\mathsf{\mathbf{A}}^{-1} - \mathsf{\mathbf{S}}\| \leq \epsilon.$$

Then an approximate solution can be found by simply evaluating

$$\mathsf{\mathbf{u}}_{\rm approx} = \mathsf{\mathbf{S}}\mathsf{\mathbf{b}}.$$

Observe that even though $\mathsf{\mathbf{A}}$ is typically sparse, the matrix $\mathsf{\mathbf{S}}$ is almost always dense. However, when $\mathsf{\mathbf{A}}$ results from the discretization of an elliptic PDE, it turns out that $\mathsf{\mathbf{S}}$ has internal structure that allows us to build, store, and apply it in a computationally efficient manner. For example, often the off-diagonal blocks are of low numerical rank, and by representing them using their thin factors, they can be efficiently stored and manipulated. The computational profile of a direct solver is typically that in situations where an iterative solver for (1.3) converges rapidly, the cost to build $\mathsf{\mathbf{S}}$ is a bit higher than the cost for a single solve using the iterative method. Once the approximate inverse $\mathsf{\mathbf{S}}$ has been built, however, it can typically be applied *very* rapidly. Advantages of direct solvers include:

(1) Acceleration in environments where a large number of solves are required for a fixed operator. Since we compute a direct solution operator $\mathsf{\mathbf{S}}$ it is simple to build this solution operator once and then apply it to all right-hand sides. An iterative solver would need to execute the entire algorithm for every right hand side. A couple orders of magnitude speed-up is often possible [29].

(2) The ability to solve problems that are intractable to iterative methods. This includes things like scattering problems near resonant frequencies and ill-conditioning due to geometry, corners, and cusps [15, 12].

(3) Direct solvers make it straight-forwards to couple multi-physics problems, or situations where different discretization techniques are used in different parts of a domain. In situations such as these, it has proven challenging to pre-condition the linear systems to attain reasonable convergence.

(4) Direct solvers of the kind considered here are often easier to implement efficiently in parallel computing environments [32].

Disadvantages of direct solvers include that they require substantially more memory than iterative solvers, and that they can be slower than the best iterative solvers in situations where good preconditioners that ensure fast convergence are available and only a single solve is required.

Remark 1 The boundary value problem in Equation (1.2) is stated to be in two dimensions. The extension to problems in three dimensions is in principle straight-forwards, as was demonstrated in [23]. However, the coding does get substantially more complex, and more sophisticated linear algebraic techniques are required to attain high practical speed [16]. All numerical examples presented in the dissertation concern the two dimensional case, but work to extend all techniques to 3D are currently in progress.

1.2 Related work

There is a rich history of fast direct solvers that motivates the work presented here. Gaussian elimination, LU factorization, and matrix inversion are all examples of direct methods that solve a matrix equation and the idea of a direct solver can be seen in many other well known algorithms. In this section, we will review variations of these ideas as they apply to solve linear PDEs.

1.2.1 Nested dissection and the multifrontal method

The work in this dissertation is conceptually related to the classical nested dissection and multifrontal methods [8, 9, 14]. The classical nested dissection method is an efficient tool for building an LU factorization of the coefficient matrix A in (1.3) that results from discretizing an elliptic PDE. This method exploits the sparsity pattern in A to minimize the fill-in that occurs when an LU factorization is computed. These methods follow the standard pattern of direct solvers in that they involve a relatively expensive "build" or "factorization" stage where the triangular factors are computed, and then admit a very fast "solve" that involves two triangular solves for any given data function. The asymptotic complexity of these solvers depend strongly on the dimension of the underlying problem. It has been demonstrated that for a typical grid, one can attain following complexities:

	Build (factorization) stage	Solve stage
1D	$O(N)$	$O(N)$
2D	$O(N^{1.5})$	$O(N \log N)$
3D	$O(N^2)$	$O(N^{4/3})$

From a practical point of view, these solvers work very well for 1D and 2D problems. For problems in 3D they provide powerful tools for problems of moderate size, but due to the $O(N^2)$ cost, and high storage requirements, they struggle to handle truly large scale problems.

1.2.2 Linear complexity algorithms for global operators

The fast multipole method (FMM) can be viewed as an early example of a direct solver with linear complexity. To illustrate, let us consider a global Poisson equation

$$-\Delta u(\boldsymbol{x}) = g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \mathbb{R}^d, \tag{1.4}$$

coupled with appropriate decay conditions at infinity to ensure well-posedness. The solution to (1.4) can be written analytically as

$$u(\boldsymbol{x}) = \int_{\mathbb{R}^d} \phi(\boldsymbol{x} - \boldsymbol{y}) g(\boldsymbol{y})\, dyy, \qquad \boldsymbol{x} \in \mathbb{R}^d, \tag{1.5}$$

where ϕ is the fundamental solution of the Laplace operator (so that, e.g., $\phi(\boldsymbol{x}) = -(2\pi)^{-1} \log |\boldsymbol{x}|$ in two dimensions). Let us suppose that g has compact support, and that we apply a quadrature rule to discretize (1.5) into a matrix-vector multiplication formula

$$\mathsf{u} = \mathsf{A}\mathsf{g}, \tag{1.6}$$

where $\mathbf{A}$ is a matrix of size $N \times N$. The Fast Multipole Method (FMM) [19] enables the matrix-vector multiplication in (1.6) to be executed to high precision in $O(N)$ time. The key idea is to exploit that the matrix $\mathbf{A}$ can be tessellated into $O(N)$ blocks in such a way that all blocks have low numerical rank, and can be applied rapidly.

An important fact about the FMM is that while it in principle has linear complexity for any dimension, the practical speed deteriorates rapidly as the dimension increases. It is extremely fast for problems in 1D [31], it is fast for 2D problems, but in 3D, additional machinery [20] is needed to attain high practical speed unless the required accuracy is very low.

The work in this dissertation is only loosely connected to the FMM. However, the FMM is one of the earlier fast methods in the computational math community and the ideas are used throughout fast direct solvers, thus laying the foundation for the work done in this dissertation.

1.2.3 Linear complexity nested dissection methods

An exciting recent development has been a combination of the ideas to exploit sparsity in nested dissection methods (cf. Section 1.2.1) with fast algorithms for dense matrices described in Section 1.2.2 to attain direct solvers that have linear complexity for all steps. The key observation is that what causes the performance of nested dissection methods to deteriorate as problem sizes increase is that they rely on dense LU factorizations of certain dense matrices that get larger and larger the more degrees of freedom are used. However, these dense matrices have internal structure that is very similar to the structures exploited in the FMM, and that makes them amenable to generalization of the FMM such as the $\mathcal{H}$-matrices of Hackbusch and co-workers [21, 22, 3, 4], or the Hierarchically Block Separable (HBS) format [17, 39]. Linear complexity algorithms that result from this combination include [40, 16, 18, 1].

1.2.4 Reducing the effective dimension of a problem

We have seen that the practical efficiency of many of the solvers discussed deteriorate rapidly as the dimensionality of the problem increases. One very effective way to reduce the dimensionality of an elliptic problem is to use a method like a boundary integral equation where a typical boundary value problem is recast as an integral equation over the boundary. For example, suppose we wish to solve the Laplace equation on a domain Ω with boundary data $h(\mathbf{x})$ on the boundary Γ. We can solve

$$\begin{cases} -\Delta u(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = h(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma, \end{cases} \tag{1.7}$$

by recasting the PDE as an integral over the boundary

$$u(\boldsymbol{x}) = \int_\Gamma \frac{\mathbf{n}(\mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})}{2\pi |\boldsymbol{x} - \boldsymbol{y}|^2} \sigma(\boldsymbol{y}) ds(\boldsymbol{y}). \tag{1.8}$$

The function $\sigma(\boldsymbol{y})$ can be found by discretizing the following integral over the boundary and solving a system of linear equations.

$$h(\boldsymbol{x}) = \frac{1}{2}\sigma(\boldsymbol{x}) + \int_\Gamma \frac{\mathbf{n}(\mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})}{2\pi |\boldsymbol{x} - \boldsymbol{y}|^2} \sigma(\boldsymbol{y}) ds(\boldsymbol{y}), \qquad \boldsymbol{x} \in \Gamma. \tag{1.9}$$

This formulation leads to well-conditioned linear systems and has the advantage of a reduction in dimension since we are now solving an equation on the boundary of the domain.

1.2.5 The Hierarchical Poincaré-Steklov (HPS) scheme

One way to explain how classical nested dissection and multifrontal methods attain high efficiency is that they too in a sense provide a way to reduce the effective dimensionality of the problem. To illustrate this idea, suppose that we are given a computational domain in the shape of a square. One way to set up the nested dissection method is that we tessellate the box into a hierarchical tree of smaller boxes, cf. Figure 1.1. In the “build stage”, the solver then executes a pass through the tree, going from smaller boxes to larger boxes. At each stage, all internal degrees of freedom in the box are eliminated, effectively reducing the local equations to a global problem defined on the boundary of the box alone. This elimination is sometimes referred to as “static condensation” in the finite element community, and has the drawback that in reducing the dimensionality of a problem, it unfortunately also replaces a sparse coefficient matrix by a dense coefficient matrix. [1]

The previously mentioned *Hierarchical Poincaré-Steklov (HPS)* scheme is a way to build a nested-dissection type solver by taking an explicitly geometric view. The dissertation research relies crucially on these solvers that were introduced by Martinsson in [29, 28], and later refined by Martinsson, Gillman, and Hao in [15, 16, 23]. We describe the HPS scheme in detail inside the dissertation. For now, let us simply mention that it has two essential advantages. The first is that it allows for very high order local discretizations to be used without deterioration in the solution speed. This is in sharp contrast to what would happen when a standard finite element or finite difference method is combined with a nested dissection solver [16]. The second advantage is that it works particularly well in combination with the structured matrix algebra techniques we described in Section 1.2.3, as demonstrated in [16] and [32].

The discretization scheme we use in the HPS scheme is related to earlier work on spectral collocation methods on composite ("multi-domain") grids, such as, e.g., [27, 41], and in particular Pfeiffer *et al* [33]. These methods are sometimes referred to as "patching methods", see e.g. [5, Ch. 5.13]. The differences and similarities between the various techniques is discussed in detail in [29].

---

[1] This procedure is also conceptually related to so-called "reduction to the interface" methods, see [26] and the references therein. Such "interface" methods also use local solution operators defined on boundaries but typically rely on variational formulations of the PDE, rather than the collocation techniques that we employ.

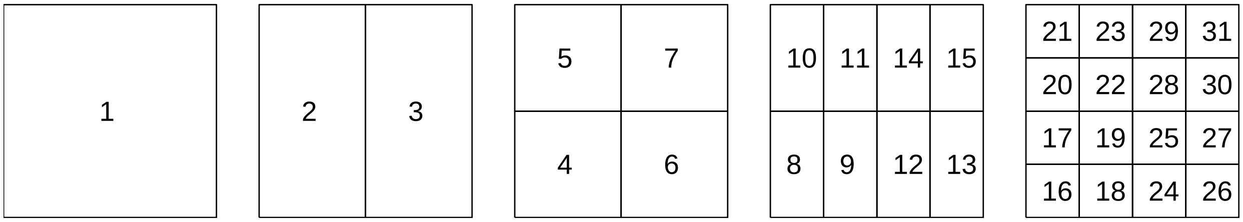


Figure 1.1: A square domain Ω is split into 4×4 leaf boxes. These are then gathered into a binary tree of successively larger boxes as described in Section 1.2.5.

1.2.6 Connections to iterative solvers

Iterative methods for solving linear systems of equations has a long history and has, by now, matured into becoming textbook material, see e.g. [35]. For elliptic equations similar to the Laplace equation that have non-oscillatory solutions (Stokes, Yukawa, etc.), multigrid methods, [38], and domain decomposition methods, [36], have been highly successful both in terms of analysis and practical implementation. For example, multigrid techniques are the basis for large scale parallel solver packages like HYPRE[2] and TRILINOS-ML[3] . Due to their low setup cost and reliable convergence for large classes of elliptic problems iterative solvers may be faster than the direct solvers we advocate here when considering single solves.

In the applications we target we require multiple solves for problems with highly oscillatory solutions, like the Helmholtz equation. For such problems it may be difficult to attain the same convergence properties of iterative methods as for elliptic problems, see e.g. [12]. For this class of problems the state of the art iterative solvers are methods based on the so called sweeping preconditioners introduced by Engquist and Ying [10, 11]. A good overview of iterative solvers for the Helmholtz problem can be found in [42].

1.3 Overview of Dissertation and Contributions

Let us briefly summarize the key contributions of the proposed dissertation research:

(1) *An extension of the HPS scheme to allow for local refinement:* It is often the case in scientific computing that there is a local loss of regularity in the solution. It is not efficient to uniformly increase the number of discretization nodes across the whole domain and we look to perform local refinement at the troublesome location. The discretization scheme used in the HPS algorithm is based on equating Neumann data across adjacent subdomains, but refinement results in the spatial discretization nodes not aligning. This can be fixed by forming interpolation operators mapping between the differing sets of nodes. This work

---

[2] `https://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods`
[3] `https://trilinos.org/packages/ml/`

has been published [2]. Section 3.6 provides more details.

(2) *An extension of the HPS scheme to allow for body loads:* The original HPS algorithm as introduced in [29, 28] does not include a body load, and we have at this point developed a modified version of HPS that does allow for body loads. The algorithmic complexity is the same as for the original scheme, although the memory complexity can be much larger if all of the solution operators are stored. We have investigated different techniques for storing only partial solution operators to mitigate the difficulties of excessive storage requirements. This work has been published [2]. Chapter 3 provides more details.

(3) *A new method for solving certain hyperbolic problems numerically.* The addition of body loads to the HPS algorithm allows us to utilize it for solving certain time dependent PDEs where a time-stepping scheme that requires an elliptic solve is used. For hyperbolic problems we can use a rational approximation for the time evolution operator and then apply the HPS algorithm to the rational approximation. This work has been published [24]. See Chapter 2 for details.

(4) *Using the HPS scheme to accelerate implicit time-stepping schemes for parabolic problems.* The standard approach for solving a parabolic equation is to use implicit time-stepping schemes that require an elliptic solve at each time-step. Our extension of the HPS scheme to allow for body loads makes it a very attractive candidate for an elliptic solver. In situations where the elliptic solve is the same at every time-step, we can exploit the fact that the solve is very fast, and amortize the relatively high cost of building the solution operator over all the time-steps. We have investigated the implementation of Explicit, Singly Diagonally Implicit Runge-Kutta (ESDIRK) and Additive Runge-Kutta (ARK) methods combined with the HPS scheme. ESDIRK methods allow for high-order stiff accuracy and L-stability while only requiring the solution to a sequence of equations which all have the same elliptic operator. This means we only have to compute and store solution operators for one elliptic operator, which is crucial since the biggest drawback of the HPS scheme is the memory

requirement. This material is currently in review. Chapter 4 provides details.

1.4 Future work

Looking forwards, a natural next target for the techniques developed in the dissertation research is to accelerate time-stepping for low Reynolds number Navier-Stokes equations. Due to the complicated flow patterns, the ability to use high-order discretizations is very valuable. Moreover, in situations where the Navier-Stokes equations are discretized using an implicit method in time, and the geometry is fixed as the simulation progresses, a sequence of elliptic equations involving the same operator needs to be solved. (We envision a situation where the non-linear component of the operator can be considered as a forcing term on the "right-hand side" of the equation.) This is an ideal set-up for a direct solver, since the cost of building the solution operator can be amortized over a large number of time-steps.

The direct solvers used in this dissertation have high practical speed in both the build and the solve stages for realistic problem sizes. However, as N increases, the methods currently used in the solve stage do have asymptotic complexity $O(N^{1.5})$. It would be a straight-forward matter to replace the current solvers by the $O(N)$ complexity methods described in [16]. However, for 2D problems, very large problem sizes would be required for this added complexity to provide any substantial benefit. In 3D the situation is quite different. The methods presented in this dissertation can without conceptual difficulty be directly implemented in 3D, and the resulting codes would work well for problems of modest size. However, for 3D problems, the $O(N^2)$ complexity of the build stage will very quickly start to hurt as N increases, and in order to solve realistic problems, rank-structured matrix algebra in the spirit of [40, 16, 18, 1] will have to be used. Building efficient machinery for this is research in progress in other research groups, with very promising initial results appearing in [32].

Chapter 2

A High-Order Scheme for Solving Wave Propagation Problems Via the Direct Construction of an Approximate Time-Evolution Operator

Note: The material in this chapter has appeared in press as: T. S. Haut. T. Babb. P. G. Martinsson. B. A. Wingate, “A High-Order Scheme for Solving Wave Propagation Problems Via the Direct Construction of an Approximate Time-Evolution Operator”, *IMA Journal of Numerical Analysis*, **36**(2), 2016. [24].

Abstract: The manuscript presents a technique for efficiently solving the classical wave equation, the shallow water equations, and, more generally, equations of the form $\partial u/\partial t = \mathcal{L}u$, where $\mathcal{L}$ is a skew-Hermitian differential operator. The idea is to explicitly construct an approximation to the time-evolution operator $\exp(\tau\mathcal{L})$ for a relatively large time-step τ. Recently developed techniques for approximating oscillatory scalar functions by rational functions, and accelerated algorithms for computing functions of discretized differential operators are exploited. Principal advantages of the proposed method include: stability even for large time-steps, the possibility to parallelize in time over many characteristic wavelengths, and large speed-ups over existing methods in situations where simulation over long times are required

Numerical examples involving the 2D rotating shallow water equations and the 2D wave equation in an inhomogenous medium are presented, and the method is compared to the 4th order Runge-Kutta (RK4) method and to the use of Chebyshev polynomials. The new method achieved high accuracy over long time intervals, and with speeds that are orders of magnitude faster than both RK4 and the use of Chebyshev polynomials.

2.1 Introduction

2.1.1 Problem formulation

We present a technique for solving a class of linear hyperbolic problems

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t}(\mathbf{x},t) = \mathcal{L}\mathbf{u}(\mathbf{x},t), & \mathbf{x} \in \Omega, \quad t > 0, \\ \mathbf{u}(\mathbf{x},0) = \mathbf{u}_0\left(\mathbf{x}\right) & \mathbf{x} \in \Omega. \end{cases} \tag{2.1}$$

Here $\mathbf{u}$ is a possibly vector valued function and $\mathcal{L}$ is a skew-Hermitian differential operator (see the end of this Section for the method's scope). The technique is demonstrated on the 2D rotating shallow water equations, as well as the variable coefficient wave equation.

The basic approach is classical, and involves the construction of a rational approximation to the time evolution operator $\exp(\tau\mathcal{L})$ in the form

$$\exp(\tau\mathcal{L}) \approx \sum_{m=-M}^{M} b_m \left(\tau\mathcal{L} - \alpha_m\right)^{-1},$$

where the time-step τ is fixed in advance and M scales linearly in τ. Once the time-step τ has been fixed, an approximate solution at times τ, 2τ, $3\tau, \ldots$ can be obtained via repeated application of the approximate time-stepping operator, since $\exp(n\tau\mathcal{L}) = \big(\exp(\tau\mathcal{L})\big)^n$. The computational profile of the method is that it takes a moderate amount of work to construct the initial approximation to $\exp(\tau\mathcal{L})$, but once it has been built, it can be applied very rapidly, even for large τ.

The efficiency of the proposed scheme is enabled by (i) a novel method [47] for constructing near optimal rational approximations to oscillatory functions such as e^{ix} over arbitrarily long intervals, and by (ii) the development [57] of a high-order accurate and stable method for pre-computing approximations to operators of the form $(\tau\mathcal{L} - \alpha_m)^{-1}$. The near optimality of the rational approximations ensures that the number $2M + 1$ of terms needed for a given accuracy is typically much smaller than standard methods that rely on polynomial or rational approximations of $\mathcal{L}$.

The proposed scheme has several advantages over typical methods, including the absence of stability constraints on the time step τ in relation to the spatial discretization, the ability to parallelize in time over many characteristic wavelengths (in addition to any spatial parallelization), and great acceleration when integrating equation (2.1) for long times or for multiple initial conditions (e.g. when employing an exponential integrator on a nonlinear evolution equation, cf. Section 2.5). A drawback of the scheme is that it is more memory intensive than standard techniques.

We restrict the scope of this paper to when the application $(\tau\mathcal{L}-\alpha_m)^{-1}\mathbf{u}_0$ can be reduced to the solution of an elliptic-type PDE for one of the unknown variables. This situation arises in geophysical fluid applications (among others), including the rotating primitive equations that are used for climate simulations. In this context, the ability to efficiently solve (2.1) can be used to construct efficient schemes for the fully nonlinear evolution equations in the presence of time scale separation (see [53]). However, the direct solver presented in Section 2.2 is quite general, and in principle can be extended to first order linear systems of hyperbolic PDEs with little modification (though such an extension is speculative and, in particular, has not been tried).

2.1.2 Time discretization

In order to time-discretize (2.1), we fix a time-step τ (the choice of which is discussed shortly), a requested precision $\delta > 0$, and "band-width" $\Lambda \in (0,\infty)$ which specifies the spatial resolution (in effect, the scheme will accurately capture eigenmodes of $\mathcal{L}$ whose eigenvalues λ satisfy $|\lambda| \leq \Lambda$). We then use an improved version of the scheme of [47] to construct a rational function,

$$R_M(ix) = \sum_{m=-M}^{M} \frac{b_m}{(ix-\alpha_m)}, \tag{2.2}$$

such that

$$\left|e^{ix} - R_M(ix)\right| \leq \delta, \qquad x \in [-\tau\Lambda, \tau\Lambda], \tag{2.3}$$

and

$$|R_M(ix)| \leq 1, \qquad x \in \mathbb{R}. \tag{2.4}$$

It now follows from (2.3) and (2.4) that if we approximate $\exp(t\mathcal{L})$ by $R_M(\tau\mathcal{L})$, the approximation error satisfies

$$\left\| e^{\tau\mathcal{L}}\mathbf{u}_0 - \sum_{m=-M}^{M} b_m \left(\tau\mathcal{L}-\alpha_m\right)^{-1}\mathbf{u}_0 \right\| \leq \delta \left\|\mathbf{u}_0\right\| + 2\left\|\mathbf{u}_0 - \mathcal{P}_\Lambda \mathbf{u}_0\right\|, \tag{2.5}$$

where $\mathcal{P}_\Lambda$ projects functions onto the subspace spanned by eigenvectors of $\mathcal{L}$ with modulus at most Λ. Here the only property of $\mathcal{L}$ that we use is that $\mathcal{L}$ is skew-Hermitian, and hence has a complete spectral decomposition with a purely imaginary spectrum.

The bound (2.4) ensures that the repeated application of $R_M(\tau\mathcal{L})$ is stable on the entire imaginary axis. It also turns out that the number $2M+1$ of terms needed in the rational approximation in (2.3) is close to optimally small (for the given accuracy δ).

The scheme described above allows a great deal of freedom in the choice of the time step τ. While classical methods typically require the time step to be a small fraction of the characteristic wavelength, we have freedom to let τ cover a large number of characteristic wavelengths. Therefore, the scheme is well suited to parallelization in time, since all the inverse operators in the approximation of the operator exponential can be applied independently. In fact, the only constraint on the size of τ is on the memory available to store the representations of the inverse operators (as explained in Section 2.1.3, the memory required for each inverse scales linearly in the number of spatial discretization parameters, up to a logarithmic factor).

2.1.3 Pre-computation of rational functions of $\mathcal{L}$

The time discretization technique described in Section 2.1.2 requires us to build explicit approximations to differential operators on the domain Ω such as $(\tau\mathcal{L} - \alpha_m)^{-1}$. We do this using a variation of the technique described in [57]. A variety of different domains can be handled, but for simplicity, suppose that Ω is a rectangle. The idea is to tessellate Ω into a collection of smaller rectangles, and to put down a tensor product grid of Chebyshev nodes on each rectangle, as shown in Figure 2.1. A function is represented via tabulation on the nodes, and then $\mathcal{L}$ is discretized via standard spectral collocation techniques on each patch. The patches are glued together by enforcing continuity of both function values and normal derivatives. This discretization results in a block sparse coefficient matrix, which can rapidly be inverted via a procedure very similar to the classical nested dissection technique of George [14]. The resulting inverse is dense but "data-sparse," which is to say that it has internal structure that allows us to store and apply it efficiently.

In order to describe the computational cost of the direct solver, let N denote the number of nodes in the spatial discretization. For a problem in two dimensions, the "build stage" of the proposed scheme constructs $2M+1$ data-sparse matrices $\{\mathsf{A}_m\}_{m=-M}^{M}$ of size $N \times N$, where each A_m approximates $(\tau\mathcal{L} - \alpha_m)^{-1}$. The build stage has asymptotic cost $\mathcal{O}(M\,N^{1.5})$, and storing the matrices requires $O(M\,N\,\log(N))$ memory. The cost of applying a matrix A_m is $\mathcal{O}(N\,\log(N))$. (We remark that the cost of building the matrices $\{\mathsf{A}_m\}_{m=-M}^{M}$ can often be accelerated to optimal $\mathcal{O}(M\,N)$ complexity [16], but since the pre-factor in the $\mathcal{O}(M\,N^{1.5})$ bound is quite small, such acceleration would have negligible benefit for the problem sizes under consideration here.) Section 2.2 describes the inversion procedure in more detail.

We remark that the spatial discretization procedure we use does not explicitly enforce that the discrete operator is exactly skew-Hermitian. However, the fact that the spatial discretization is done to very high accuracy means that it is in practice very nearly so. Numerical experiments indicate that the scheme as a whole is stable in every regime where it was tested.

2.1.4 Comparison to existing approaches

The approach of using proper rational approximations for applying matrix exponentials has a long history. In the context of operators with negative spectrum (e.g. for parabolic-type PDEs), many authors have discussed how to compute efficient rational approximations to the decaying exponential e^{-x}, including using Cauchy's integral formula coupled with Talbot quadrature (cf. [63]), and optimal rational approximations via the Carathéodory-Fejer method (cf. [63]) or the Remez algorithm [45]. However, such methods are less effective (or not applicable) when applied to approximating oscillatory functions such as e^{ix} over long intervals. For computing functions of parabolic-type linear operators, the approach of combining rational approximations and compressed representations of the solution operators using so-called $\mathcal{H}$-matrices has been proposed in [50].

Common approaches for applying the exponential of skew-Hermitian operators include high-order time-stepping methods, scaling-and-squaring coupled with Padé approximations (cf. [54]) or Chebyshev polynomials (cf. [43]), and polynomial or rational Krylov methods (cf. [55] and [51]).

All these methods iteratively build up rational or polynomial approximations to the operator exponential, and correspondingly approximate the spectrum $e^{i\omega_n \tau}$ of $e^{\tau \mathcal{L}}$ with polynomials or rationals. Therefore, the near optimality of (2.2) and the speed of applying the inverse operators in (2.5) will generally translate into high efficiency relative to standard methods. In contrast to these standard approaches, the method proposed in this paper can also be trivially parallelized in time over many characteristic wavelengths.

In addition to approaches that rely on polynomial or rational approximations, let us mention two alternative approaches for time-stepping on wave propagation problems. The authors in [44] combine separated representations of multi-dimensional operators, partitioned low rank compressions of matrices, and (near) optimal quadrature nodes for band-limited functions, in order to compute compressed representations of the operator exponential over $1-2$ characteristic wavelengths. Along different lines, the authors in [48] use wave atoms to construct compressed representations of the (short time) operator exponential, and in particular can bypass the CFL constraint.

2.1.5 Outline of manuscript

The paper is organized as follows. In Section 2.2, we briefly describe the direct solver in [57]. We then discuss in Section 2.3 a technique for constructing efficient rational approximations of general functions, and specialize to the case of approximating the exponential e^{ix} and the phi-functions for exponential integrators [46]. In Section 2.4, we present applications of the method for both the 2D rotating shallow water equations and the 2D wave equation in inhomogenous medium. In particular, we compare the accuracy and efficiency of this approach against 4th order Runge-Kutta and the Chebyshev polynomial method (in our comparisons, we use the same spectral element discretization).

2.2 Spectral element discretization

This section describes how to efficiently compute a highly accurate approximation to the inverse operator $(\mathcal{L}-\alpha)^{-1}$, where $\mathcal{L}$ is a skew-Hermitian operator. As mentioned in the introduction, we restrict our discussion to environments where application of the inverse can be reformulated as a scalar elliptic problem. This reformulation procedure is illustrated for the classical wave equation and for the shallow water equations in Section 2.2.1. Section 2.2.2 describes a high-order multidomain spectral discretization procedure for the elliptic equation. Section 2.2.3 describes a direct solver for the system of linear equations arising upon discretization.

2.2.1 Reformulation as an elliptic problem

In many situations of practical interest, the task of solving a hyperbolic equation $(\mathcal{L}-\alpha)u = f$, where $\mathcal{L}$ is a skew-Hermitian operator, can be reformulated as an associated elliptic problem. In this section, we illustrate the idea via two representative examples. Example 1 is of particular relevance to geophysical fluid applications, which serve as a major motivation of this algorithm.

Example 1 — the shallow water equation: We consider the rotating shallow water equations,

$$\begin{aligned} \mathbf{v}_t &= -fJ\mathbf{v} + \nabla\eta, \\ \eta_t &= \nabla\cdot\mathbf{v}, \end{aligned} \tag{2.6}$$

where $\mathbf{v}(\mathbf{x}) = (v_1(\mathbf{x}), v_2(\mathbf{x}))$ denotes the fluid velocity, $\eta(\mathbf{x})$ denotes perturbed surface elevation, f is the (possibly spatially varying) Coriolis frequency, and

$$J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

On the sphere, $f = 2\Omega\sin\phi$; on the plane, f is constant. We write system (2.6) in the form

$$\mathbf{u}_t = \mathcal{L}\mathbf{u},$$

where

$$\mathcal{L}\begin{pmatrix} \mathbf{v} \\ \eta \end{pmatrix} = \begin{pmatrix} -fJ\mathbf{v} + \nabla\eta \\ \nabla\cdot\mathbf{v} \end{pmatrix}. \tag{2.7}$$

Although we only consider the case when the Coriolis frequency is constant, the method generalizes to non-constant coefficient f (see also the example in the next section) and is of particular relevance for a spectral element discretization on the cubed sphere.

In order to apply the method in this paper, we use the standard fact (cf. [62]) that if

$$(\mathcal{L}-\alpha)\begin{pmatrix}\mathbf{v}\\ \eta\end{pmatrix}=\begin{pmatrix}\mathbf{v_0}\\ \eta_0\end{pmatrix}, \tag{2.8}$$

then η satisfies the elliptic equation

$$\nabla\cdot\left(\mathcal{A}_\alpha\nabla\eta\right)-\alpha\eta=\eta_0+H\nabla\cdot\mathcal{A}_\alpha\mathbf{v_0}. \tag{2.9}$$

Here $\mathcal{A}_\alpha$ is defined by

$$\mathcal{A}_\alpha=\frac{1}{\alpha^2+f^2}\begin{pmatrix}\alpha & f\\ -f & \alpha\end{pmatrix}.$$

Once η is computed, $\mathbf{v}$ can be obtained directly,

$$\mathbf{v}=-\mathcal{A}_\alpha\mathbf{v_0}+\mathcal{A}_\alpha\nabla\eta. \tag{2.10}$$

When f is constant, equation (2.9) reduces to

$$\left(\Delta-\frac{\alpha^2+f^2}{c^2}\right)\eta=\frac{\alpha^2+f^2}{c^2\alpha}\left(\eta_0+H\nabla\cdot\left(\mathcal{A}_\alpha\mathbf{v_0}\right)\right). \tag{2.11}$$

Example 2 — the wave equation: Consider the wave propagation problem

$$u_{tt}=\kappa\Delta u,\quad \mathbf{x}\in[0,1]\times[0,1], \tag{2.12}$$

where $\kappa(\mathbf{x})>0$ is a smooth function, the initial conditions $u(\mathbf{x},0)$ and $u_t(\mathbf{x},0)$ are prescribed, and periodic boundary conditions are used.

In order to apply the method in this paper, we reformulate (2.12) as a first order system in both time and space by defining $v=u_t$, $w=u_x$, and $z=u_y$. Then we have that

$$\begin{pmatrix}w_t\\ z_t\\ v_t\end{pmatrix}=\begin{pmatrix}0 & 0 & \partial_x\\ 0 & 0 & \partial_y\\ \kappa\partial_x & \kappa\partial_y & 0\end{pmatrix}\begin{pmatrix}w\\ z\\ v\end{pmatrix}, \tag{2.13}$$

with initial conditions

$$v(\mathbf{x},0)=u_0(\mathbf{x}),\ \ w(\mathbf{x},0)=\frac{\partial u_0}{\partial x}(\mathbf{x}),\ \ z(\mathbf{x},0)=\frac{\partial u_0}{\partial y}(\mathbf{x}).$$

Here the scalar function u to the original system (2.12) can be recovered after the final time step by solving the elliptic equation $\Delta u = w_x + z_y$.

To apply the method in this paper, we compute the solution to

$$(\mathcal{L} - \alpha) \begin{pmatrix} w \\ z \\ v \end{pmatrix} = \begin{pmatrix} v_x - \alpha w \\ v_y - \alpha z \\ \kappa \left(w_x + z_y \right) - \alpha v \end{pmatrix} = \begin{pmatrix} w_0 \\ z_0 \\ v_0 \end{pmatrix} \tag{2.14}$$

as follows. First, solving for w and z in terms of v,

$$w = \frac{1}{\alpha} \left(v_x - w_0 \right), \quad z = \frac{1}{\alpha} \left(v_y - z_0 \right), \tag{2.15}$$

it is straightfoward to show that

$$\left(\Delta - \alpha^2 \kappa^{-1} \right) v = \alpha \kappa^{-1} v_0 + \frac{\partial w_0}{\partial x} + \frac{\partial z_0}{\partial y}. \tag{2.16}$$

Once v is known, w and z can then be computed directly via (2.15).

2.2.2 Discretization

In this section, we describe a high-order accurate discretization scheme for elliptic boundary value problems such as (2.11) and (2.16) which arise in the solution of hyperbolic evolution equations. Specifically, we describe the solver for a boundary value problem (BVP) of the form

$$\mathcal{B} u \left(\mathbf{x} \right) = f \left(\mathbf{x} \right), \ \ \mathbf{x} \in \Omega, \tag{2.17}$$

where $\mathcal{B}$ is an elliptic differential operator. To keep things simple, we consider only square domains $\Omega = [0, 1]^2$, but the solver can easily be generalized to other domains. The solver we use is described in detail in [58], our aim here is merely to give a high-level conceptual description.

The PDE (2.17) is discretized using a multidomain spectral collocation method. Specifically, we split the square Ω into a large number of smaller squares (or rectangles), and then put down a tensor product grid of $p \times p$ Chebyshev nodes on each small square, see Figure 2.1. The parameter p is chosen so that dense computations involving matrices of size $p^2 \times p^2$ are cheap ($p = 20$ is often a good choice). Let $\{\mathbf{x}_j\}_{j=1}^N$ denote the total set of nodes. Our approximation to the solution u of (2.17) is then represented by a vector $\mathbf{u} \in \mathbb{C}^N$, where the j'th entry is simply an approximation to the function value at node $\mathbf{x}_j$, so that $\mathbf{u}(j) \approx u(\mathbf{x}_j)$. The discrete approximation to (2.17) then takes the form

$$\mathbf{B}\mathbf{u} = \mathbf{f}, \tag{2.18}$$

where $\mathbf{B}$ is an $N \times N$ matrix. The j'th row of (2.18) is associated with a collocation condition for node $\mathbf{x}_j$. For all j for which $\mathbf{x}_j$ is a node in the *interior* of a small square (filled circles in Figure 2.1), we directly enforce (2.17) by replacing all differentiation operators by spectral differentiation operators on the local $p \times p$ tensor product grid. For all j for which $\mathbf{x}_j$ lies on a *boundary* between two squares (hollow squares in Figure 2.1), we enforce that normal fluxes across the boundary are continuous, where the fluxes from each side of the boundary are evaluated via spectral differentiation on the two patches (corner nodes need special treatment, see [58]).

2.2.3 Direct solver

The discrete linear system (2.18) arising from discretization of (2.17) is block-sparse. Since it has the typical sparsity pattern of a matrix discretizing a 2D differential operator, it is possible to compute its LU factorization in $O(N^{1.5})$ operations using a nested dissection ordering of the nodes [49, 14] that minimizes fill-in. Once the LU-factors have been computed, the cost of a linear solve is $O(N \log N)$. In the numerical computations presented in Section 2.4, we use a slight variation of the nested-dissection algorithm that was introduced in [57] for the case of homogeneous equations. The extension to the situation involving body loads is straight-forward, see [58].

We note that by exploiting internal structure in the dense sub-matrices that appear in the factors of $\mathbf{B}$ as the factorization proceeds, the complexity of both the factorization and the solve stages can often be reduced to optimal $O(N)$ complexity [16]. However, for the problem sizes considered in this manuscript, there would be little practical gain to implementing this more complex algorithm.

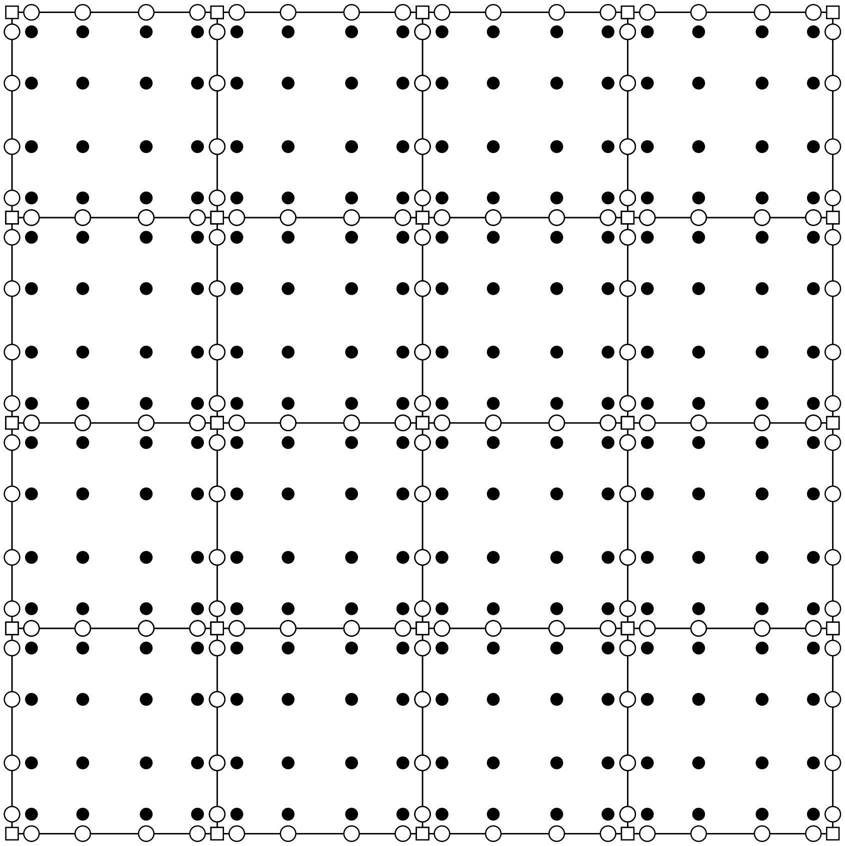


Figure 2.1: Illustration of the grid of points $\{\mathbf{x}_j\}_{j=1}^N$ introduced to discretize (2.17) in Section 2.2.2. The figure shows a simplified case involving 4×4 squares, each holding a 6×6 local tensor product grid of Chebyshev nodes. The PDE (2.17) is enforced via collocation using spectral differentiation on each small square at all solid ("internal") nodes. At the hollow ("boundary") nodes, continuity of normal fluxes is enforced.

2.3 constructing rational approximations

We now discuss how to construct efficient rational approximations to general smooth functions $f(x)$. For concreteness, we consider approximating the phi functions

$$\varphi_0(x) = e^{ix}, \;\; \varphi_1(x) = \frac{e^{ix}-1}{ix}, \;\; \varphi_2(x) = \frac{e^{ix}-ix-1}{ix^2},$$

that arise for high-order exponential integrators (cf. [63]). By considering the real and imaginary components separately, we assume that $f(x)$ is real-valued (it turns out that the poles in the approximation will be the same for the real and imaginary components).

The construction proceeds in two steps; the second step is actually a pre-computation and need only be done once, but is presented last for clarity. First, we construct an approximation to $f(x)$ by sums of shifted Gaussians $\psi_h(x) = (4\pi)^{-1/2} e^{-x^2/(4h^2)}$ (see Section 2.3.1 for details),

$$\left| f(x) - \sum_{-M}^{M} b_m \psi_h(x+nh) \right| \le \delta_1, \quad -\Lambda \le x \le \Lambda. \tag{2.19}$$

Here h is inversely proportional to the bandlimit of $f(x)$, and M controls the interval Λ over which the approximation is valid (roughly $|x| \lesssim Mh$). When $f(x) = e^{ix}$, the coefficients are explicitly given by $c_m = \left(\widehat{\psi_h}(1)/h\right) e^{-2\pi inh}$, and the approximation is remarkably accurate (see 2.23 for error bounds). Second, using the approach in [47], a rational approximation to $\psi_1(x) = (4\pi)^{-1/2} e^{-x^2/4}$ is constructed over the real line (see Section 2.3.3 for details),

$$\left| \psi_1(x) - 2\mathrm{Re}\left(\sum_{j=-L}^{L} \frac{a_j}{ix - (\mu + ij)} \right) \right| \le \delta_2, \quad x \in \mathbb{R}. \tag{2.20}$$

Notice that the imaginary parts of the poles in the above approximation are integer multiples $j = 0, \pm 1, \dots, \pm L$. For $L = 11$, we construct μ and coefficients a_j such that the L^∞ approximation error δ_2 satisfies $\delta_2 < 10^{-12}$ (see Table 2.1). Finally, combining (2.19) and (2.20), we obtain a rational approximation to $f(x)$,

$$\left| f(x) - 2\mathrm{Re}\left(\sum_{n=-M-L}^{M+L} \frac{c_n}{ix - h(\mu + in)} \right) \right| \le \delta_1 + 2(M+L)\delta_2.$$

Here the coefficients c_n are given by

$$c_n = h \sum_{k=L_1}^{L_2} a_k b_{n-k},$$

where

$$L_1(n) = \max(-L, n-M), \quad L_2(n) = \max(-L, n-M).$$

Importantly, constructing the rational approximation (2.20) to $\psi(x)$ need only be done once. In particular, once μ and the coefficients a_j are pre-computed, rational approximations to general functions $f(x)$ over arbitrarily long spatial intervals can be obtained with minimal effort, as discussed in Section 2.3.1. We present μ, and the coefficients a_j, $j = -11, \ldots, 11$, in Table 2.1, which are sufficient to yield an L^∞ error $\delta_1 \approx 7 \times 10^{-13}$ in (2.20) .

Using the reduction algorithm in [52], we find that the rational approximation constructed for e^{ix} is close to optimal in the L^∞ norm, for a given accuracy δ and spatial cutoff A. In fact, the construction in this paper uses only 1.2 times more poles than the near optimal rational approximation obtained from [52] (when $\delta = 10^{-10}$ and $A = 56\pi$, which we use in our numerical experiments). We note that the residues corresponding to this near optimal approximation can be very large and, for this reason, we prefer to use the sub-optimal approximation instead.

As clarified in Sections 2.3.1 and 2.3.3, the same poles can be used to approximate multiple functions with the same bandlimit. For example, we can use the same poles to approximate all functions $e^{2\pi itx}$, for $0 \le t \le 1$, since all these functions have bandlimit less than or equal to $e^{2\pi ix}$; the dependence on t is only through the coefficients, which are given explicitly by $c_m = \left(\widehat{\psi_h}(t)/h\right) e^{-2\pi inth}$. In particular, the poles $\alpha_m = h(\mu + im)$ are independent of t and yield uniformly accurate approximations to e^{itx} on the same interval $[-\Lambda, \Lambda]$. This observation enables the efficient computation of multiple operator exponentials $e^{s_k \mathcal{L}} \mathbf{u}_0$, for $s_k = tk/L$, using the same computed solutions $(t\mathcal{L} - \alpha_m)^{-1} \mathbf{u}_0$, $m = 1, \ldots, M$. A similar comment applies to the phi-functions from exponential integrators.

Generally, any rational approximation to e^{ix} (or more general functions) must share the same number of zeros within the interval of interest; in particular, since the rational approximation can be expressed as a quotient of polynomials, it is therefore subject to the Nyquist constraint. However, one advantage of this approximation method is that it allows efficient rational approximations of functions that are spatially localized. In fact, since the approximation (2.19) involves highly localized Gaussians, the subsequent rational approximations are able to represent spatially localized functions as well as highly oscillatory functions using (perhaps a subset) of the same collection of poles. This allows the ability to take advantage of spectral gaps (e.g. from scale separation between fast and slow waves) and possibly bypass the Nyquist constraint under certain circumstances.

2.3.1 Gaussian approximations to a general function

We discuss how to construct the approximation (2.19). To do so, we choose h small enough that the function $\hat{f}(\xi)$ is zero (or approximately so) outside the interval $[-1/(2h), 1/(2h)]$. Then we can expand $\hat{f}(\xi)/\widehat{\psi_h}(\xi)$ in a Fourier series,

$$\frac{\hat{f}(\xi)}{\widehat{\psi_h}(\xi)} = \sum_{-\infty}^{\infty} c_m e^{2\pi i m h \xi}, \tag{2.21}$$

where

$$c_m = h \int_{-1/(2h)}^{1/(2h)} e^{-2\pi i m h \xi} \frac{\hat{f}(\xi)}{\widehat{\psi_h}(\xi)} d\xi.$$

Transforming (2.21) back to the spatial domain, we have that

$$f(x) = \sum_{-\infty}^{\infty} c_m \psi_h(x + mh).$$

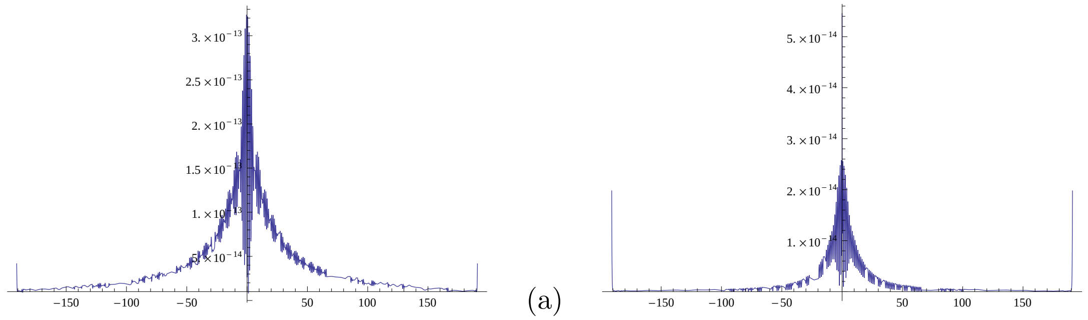
Notice that the functions $\psi_h(x + mh)$ are tighly localized in space, and truncating the above series from $-M$ to M yields accurate approximations for $-(M-b)hx < x < (M-b)hx$, where $b > 0$ is a small number that is related to the decay of $\psi_h(x)$. We remark that the authors in [59] discuss a related method of constructing quasi-interpolating representations via sums of Gaussians (see [60] for a comprehensive survey).

Specializing to the case when $f(x) = e^{2\pi i x}$, we have that $\hat{f}(\xi) = \delta(\xi - 1)$, and so the coefficients c_m are given by

$$c_m = \frac{h}{\widehat{\psi_h}(1)} e^{-2\pi i m h}. \tag{2.22}$$

Similarly, for functions $\varphi_1(x)$ and $\varphi_2(x)$, the coefficients c_m can be obtained numerically using the

Figure 2.2: The absolute error in the Gaussian approximations of $\varphi_j\left(x\right)$ for $j=1,2$ (plots (a) and (b)) , using $h=1$ and $M=200$.



(a) (a)

fact that

$$\widehat{\phi_1}\left(\xi\right)=\begin{cases}2\pi, & -\frac{1}{2\pi}\leq\xi\leq 0,\\ 0, & \text{otherwise.}\end{cases}$$

and

$$\widehat{\phi_2}\left(\xi\right)=\begin{cases}\left(2\pi\right)^2\left(\xi+\frac{1}{2\pi}\right), & -\frac{1}{2\pi}\leq\xi\leq 0,\\ 0, & \text{otherwise.}\end{cases}$$

For example, the coefficients c_m for e.g. $\phi_1\left(x\right)$ can be computed via discretization of the integral,

$$c_m=h\int_{-1/(2\pi)}^{0}e^{-2\pi imh\xi}\frac{e^{-2\pi imh\xi}}{\widehat{\psi_h}\left(\xi\right)}d\xi.$$

In Figure 2.2, we plot the error,

$$\left|\varphi_j\left(x\right)-\sum_{-\infty}^{\infty}c_{m,j}\psi\left(x+mh\right)\right|,$$

for the phi functions $\varphi_1\left(x\right)$ and $\varphi_1\left(x\right)$, where we choose $h=1$ and $M=200$; notice that the choice of h corresponds to the bandlimit of $\varphi_j\left(x\right)$. As shown in Figure 2.2, the error is smaller than $\approx 3\times 10^{-13}$ for all $-191\leq x\leq 191$, and is shown to begin to rise at the ends of the intervals, which are close to Mh. This behavior can be understood by noting that

$$\left|\varphi_j\left(x\right)-\sum_{-M}^{M}c_{m,j}\psi_1\left(x+m\right)\right|\leq\sum_{|m|>M}\left|c_{m,j}\right|\psi_1\left(x+m\right),$$

where we used that the support of $\widehat{\varphi_j}$ is contained in $[-1/2,1.2]$. Since the functions $\psi_1\left(x+m\right)$ for $m>M$ decay rapidly away from $x=-m$, the error from truncation is negligible when $|x|\leq\left(M-m_0\right)$ and $m_0=\mathcal{O}\left(1\right)$.

We remark that, for the function e^{ix}, it is shown in Section 2.3.2 that the approximation for e^{ix} satisfies

$$\left| e^{ix} - \sum_{m=-M}^{M} c_m \psi_h \left(x + mh\right) \right| \leq \frac{1}{\widehat{\psi_h}\left(1\right)} \left(\sum_{k \neq 0} \widehat{\psi_h}\left(\frac{k}{h}\right) + \sum_{|m|>M} \psi_h \left(x + mh\right) \right), \tag{2.23}$$

where c_m is defined in (2.22). We see that the first sum is negligible for e.g. $h \lesssim 1$, owing to the tight frequency localization of ψ_h. Similarly, the second sum is negligible when $|x| \leq \left(M - m_0\right) h$ and $m_0 = \mathcal{O}\left(1\right)$, owing to the tight spatial localization of ψ.

2.3.2 Error bounds

We now derive the error bound (2.23). To do so, we use the Poisson summation formula,

$$\sum_{m=-\infty}^{\infty} \Psi_h \left(x + mh\right) = \frac{1}{h} \sum_{k=-\infty}^{\infty} e^{2\pi i (k/h) x} \widehat{\Psi_h}\left(\frac{k}{h}\right).$$

Applying this to $\Psi_h\left(x\right) = e^{-2\pi i x} \psi_h\left(x\right)$, we have that

$$\begin{aligned} \sum_{m=-\infty}^{\infty} \Psi_h \left(x + mh\right) &= e^{-2\pi i x} \sum_{m=-\infty}^{\infty} e^{-2\pi i m h} \psi_h \left(x + mh\right) \\ &= \frac{1}{h} \sum_{k=-\infty}^{\infty} e^{2\pi i (k/h) x} \widehat{\Psi_h}\left(\frac{k}{h}\right) \\ &= \frac{1}{h} \sum_{k=-\infty}^{\infty} e^{2\pi i (k/h) x} \widehat{\psi_h}\left(\frac{k}{h} + 1\right), \end{aligned}$$

where the last inequality uses the fact that

$$\widehat{\Psi_h}\left(\frac{k}{h}\right) = \widehat{\psi_h}\left(\frac{k}{h} + 1\right).$$

Therefore,

$$\left| \sum_{m=-\infty}^{\infty} e^{-2\pi i m h} \psi_h \left(x + mh\right) - \frac{\widehat{\psi_h}\left(1\right)}{h} e^{2\pi i x} \right| \leq \frac{1}{h} \sum_{k \neq 0} \widehat{\psi_h}\left(\frac{k}{h}\right).$$

Finally, truncating the sum we obtain the bound (2.23).

2.3.3 Rational approximation to a Gaussian

We now discuss how to construct the approximation (2.20).

To do so, we first use AAK theory (see [47] for details) to construct a near optimal rational approximation,

$$\left| \frac{1}{\sqrt{4\pi}} e^{-x^2/4} - \mathrm{Re}\left(\sum_{j=1}^{N} \frac{b_j}{ix + \alpha_j} \right) \right| \leq \delta.$$

For an accuracy of $\delta \approx 10^{-13}$, 13 poles γ_j are required.

Setting $\mu = \min_j \mathrm{Re}\left(\alpha_j\right)$, we next look for a rational approximation to $(4\pi)^{-1/2}\, e^{-x^2/4}$ of the form

$$R\left(x\right) = \mathrm{Re}\left(\sum_{j=-L}^{L} \frac{a_j}{ix + \mu + ij} \right), \tag{2.24}$$

where we take $L = 11$. We find the coefficients a_j by minimizing the L^{∞} error

$$\left\| \frac{1}{\sqrt{4\pi}} e^{-x^2/4} - \mathrm{Re}\left(\sum_{j=-L}^{L} \frac{a_j}{ix_n + \mu + ij} \right) \right\|_{\infty},$$

where the points $x_n \in [-30, 30]$ are chosen to be more sparsely distributed outside the numerical support of $e^{-x^2/4}$; the interval $[-30, 30]$ is found experimentally to yield high accuracy for the approximation over the entire real line. Finding the coefficients a_j, $j = -L, \ldots, L$, that minimize the L^{∞} error can be cast as a convex optimization problem, and a standard algorithm can be used (we use Mathematica). The resulting approximation error is shown in Figure 2.3; the error remains less than $\approx 7 \times 10^{-13}$ for all $x \in \mathbb{R}$.

We display the real number μ, and the coefficients a_j, $j = 1, \ldots, 11$. In particular, these numbers are the only parameters that are needed in order to construct rational approximations to general functions on spatial intervals of any size.

In Figure 2.4, we show the resulting rational approximations of $\cos\left(2\pi x\right)$ and $\sin\left(2\pi x\right)$, which use the same 172 complex-conjugate pairs of poles; the L^{∞} error is seen to be $\approx 10^{-10}$ over the interval $-28 \leq x \leq 28$.

Figure 2.3: Error in the rational approximation (2.20) to $e^{-x^2/4}$

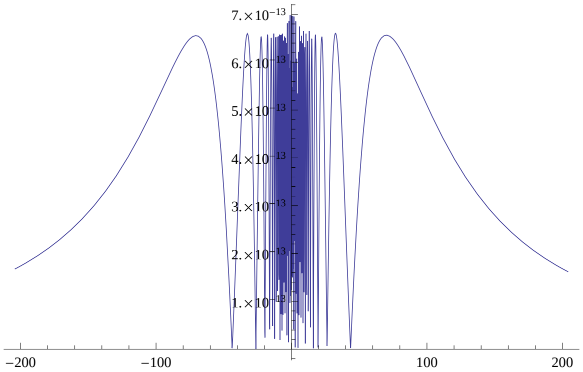
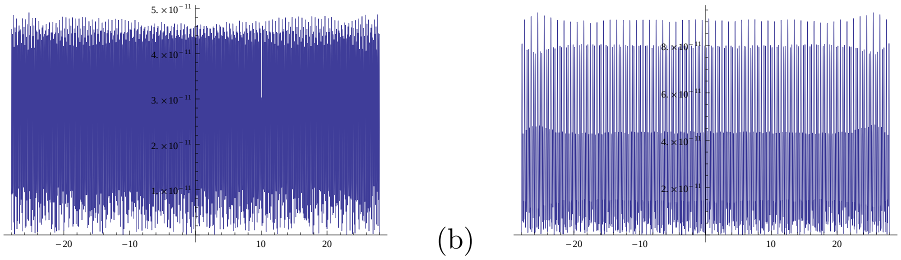


Table 2.1: Coefficients a_j, $j = -11, \ldots, 11$, and number μ, in the rational approximation (2.24).

$\mu = -4.315321510875024,$
$a_{-11} = \left(-1.0845749544592896 \times 10^{-7}, 2.77075431662228 \times 10^{-8}\right),$
$a_{-10} = \left(1.858753344202957 \times 10^{-8}, -9.105375434750162 \times 10^{-7}\right),$
$a_{-9} = \left(3.6743713227243024 \times 10^{-6}, 7.073284346322969 \times 10^{-7}\right),$
$a_{-8} = \left(-2.7990058083347696 \times 10^{-6}, 0.000011256482763934\right),$
$a_{-7} = (0.000014918577548849352, -0.000031627848676193\mathrm{2}),$
$a_{-6} = (-0.0010751767283285608, -0.00047282220513073084),$
$a_{-5} = (0.003816465653840016, 0.017839810396560574),$
$a_{-4} = (0.12124105653274578, -0.12327042473830248),$
$a_{-3} = (-0.9774980792734348, -0.1877130220537587),$
$a_{-2} = (1.3432866123333178, 3.2034715228495942),$
$a_{-1} = (4.072408546157305, -6.123755543580666),$
$a_0 = -9.442699917778205,$
$a_1 = (4.072408620272648, 6.123755841848161),$
$a_2 = (1.3432860877712938, -3.2034712658530275),$
$a_3 = (-0.9774985292598916, 0.18771238018072134),$
$a_4 = (0.1212417070363373, 0.12326987628935386),$
$a_5 = (0.0038169724770333343, -0.01783924222244388\mathrm{8}),$
$a_6 = (-0.0010756025812659208, 0.00047318749173438\mathrm{58}),$
$a_7 = (0.000014713754789095218, 0.000031358475831136815),$
$a_8 = \left(-2.659323898804944 \times 10^{-6}, -0.000011341571201752273\right),$
$a_9 = \left(3.6970377676364553 \times 10^{-6}, -6.517457477594937 \times 10^{-7}\right),$
$a_{10} = \left(3.883933649142257 \times 10^{-9}, 9.128496023863376 \times 10^{-7}\right),$
$a_{11} = \left(-1.0816457995911385 \times 10^{-7}, -2.954309729192276 \times 10^{-8}\right)$

Figure 2.4: Error in the rational approximations of $\sin(2\pi x)$ and $\cos(2\pi x)$ (plots (a) and (b)), for $-28 \le x \le 28$. These approximations use the same 172 pairs of complex-conjugate poles.



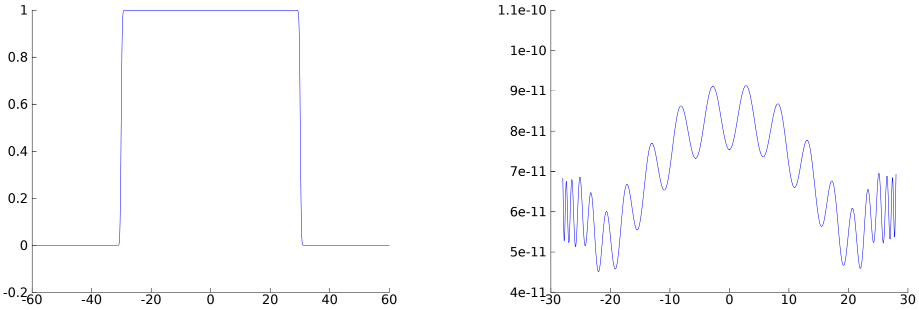
(a)

(b)

Figure 2.5: (a) Plot of the rational filter function $S(ix)$, for $-60 \le x \le 60$. (b) Plot of the difference $|S(ix) - 1|$ for $-28 \le x \le 28$.

(a)

(b)



2.3.4 Constructing rational approximation of modulus bounded by unity

For our applications, it is important that the approximation to e^{ix} is bounded by unity on the real line. In particular, the Gaussian approximation for e^{ix} constructed in Section 2.3.1 has absolute value larger than one when $|x| \approx Mh$, and this can lead to instability in repeated applications of $e^{t\mathcal{L}}$.

The basic idea is to construct a rational function $S(ix)$ that satisfies $S(ix) \approx 1$ for $|x| \lesssim M_0 h$ and $S(ix) \approx 0$ for $|x| \gtrsim M_0 h$. As long as M_0 is slightly less than M, the function $S(ix) R_M(ix)$ accurately approximates e^{ix} for $|x| \lesssim M_0 h$, and decays rapidly to zero for $|x| \gtrsim M_0 h$. Therefore, $|S(ix) R_M(ix)| \leq 1$ for all $x \in \mathbb{R}$, and repeated application of $S(t\mathcal{L}) R_M(t\mathcal{L}) \mathbf{u}_0$ is stable for all $t > 0$. In Figure 2.5, we plot rational filter that uses 33 complex-conjugate poles; we see that $|S(ix) - 1| \approx 10^{-10}$ for $-28 \leq x \leq 28$.

Although the above approach results in a stable method, we have found it more efficient to use a slightly modified version. This is motivated by the following simple observation: since $\mathbf{u}_0(\mathbf{x})$ is real-valued,

$$\overline{(t\mathcal{L} - \alpha)^{-1} \mathbf{u}_0} = (t\mathcal{L} - \overline{\alpha})^{-1} \mathbf{u}_0. \tag{2.25}$$

Recalling that the poles from Section 2.3.3 come in complex-conjugate pairs, only half the matrix inverses need to be pre-computed and applied if (2.25) is used. However, directly using (2.25) results in numerical instabilities, where small errors in the high frequencies are amplified after successive applications of $R_M(t\mathcal{L}) \mathbf{u}_0$. The fix is to eliminate the errors in the high frequency components by instead computing $S(k_0 \Delta) R_M(t\mathcal{L}) \mathbf{u}_0$, where k_0 is determined by the frequency content of $\mathbf{u}_0(\mathbf{x})$ and the operator $S(k_0 \Delta)$ only affects the highest wavenumbers. Since the transition region between $S(ix) \approx 1$ and $S(ix) \approx 0$ can be made arbitrarily small (see Figure 2.5), the operator $S(k_0 \Delta)$ behaves like a spectral projector.

We now discuss how to construct $S(ix)$. To do so, we use that (see [61])

$$\left| \frac{1}{\widehat{\psi_h}(1)} \sum_{-\infty}^{\infty} \psi_h(x + hm) - 1 \right| \leq \frac{1}{h \widehat{\psi_h}(1)} \sum_{k \neq 0} \widehat{\psi_h}\left(\frac{k}{h}\right),$$

which follows from the Poisson summation formula. For $h \lesssim 1$, the right hand side is negligible, owing to the tight frequency localization of $\widehat{\psi_h}(\xi)$. Truncating the above sum and using the tight spatial localiztion of $\psi_h(x)$, we see that the function

$$\chi(x) = \sum_{-M_0}^{M_0} \psi_h(x + mh), \tag{2.26}$$

is approximately unity for $|x| \lesssim M_0 h$, and decays to zero rapidly when $|x| \gtrsim M_0 h$. It also holds out that $|\chi(x)| \leq 1$ for all $x \in \mathbb{R}$. Therefore, using the techniques from Sections 2.3.1 and 2.3.3, we construct a rational approximation $Q(ix)$ to the function $\chi(x)$ in (2.26),

$$\left| Q(ix) - \sum_{-M_0}^{M_0} \psi_h(x + mh) \right| \leq \delta, \quad x \in \mathbb{R}, \tag{2.27}$$

The number of poles required to represent the sub-optimal approximation for $Q(x)$ can be drastically reduced with the reduction algorithm [52], which produces another proper rational function $S(x)$ such that

$$|Q(ix) - S(ix)| \leq \delta_0, \;\; x \in \mathbb{R},$$

and with a near optimally small number of poles for the prescribed L^∞ error δ_0. Since the poles of $S(ix)$ and $R(ix)$ are distinct, the function $S(ix)R(ix)$ can be expressed as a proper rational function. The final function $S(ix)$ is what is shown in Figure 2.5.

2.4 Examples

2.4.1 The 2D (rotating) shallow water equations

We apply the technique proposed to the linear shallow water equations

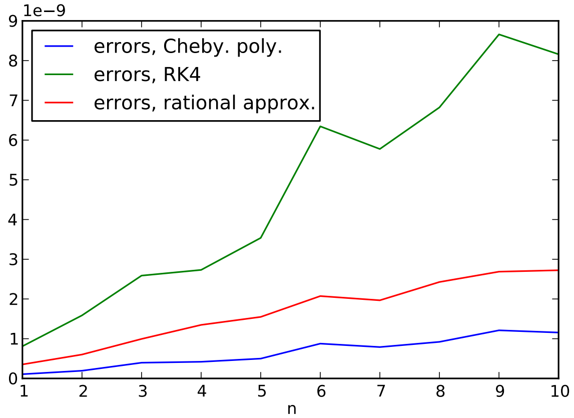
$$\begin{aligned} \mathbf{v}_t &= -fJ\mathbf{v} + \nabla\eta, \\ \eta_t &= \nabla \cdot \mathbf{v}, \end{aligned}$$

where all quantities are as in Section 2.2.1, cf. equation (2.6).

We apply the algorithm in the spatial domain $[0,1] \times [0,1]$, using periodic boundary conditions and a constant Coriolis force $f = 1$. In this case, an exact solution can be computed analytically

Figure 2.6: (a) Plots of the L^{∞} error, $\left\|\mathbf{u}_n - e^{n\tau L}\mathbf{u}_0\right\|_{\infty}$, versus the big time step $n\tau$, where $\tau = 3$ and $1 \leq n \leq 10$. Here the approximation $\mathbf{u}_n$ is computed via RK4, the Chebyshev polynomial method, and the rational approximation method. (b) Plots of the computation time (min.) versus the big time step $n\tau$, for the RK4, the Chebyshev polynomial method, and the rational approximation method.

(a)



(b)

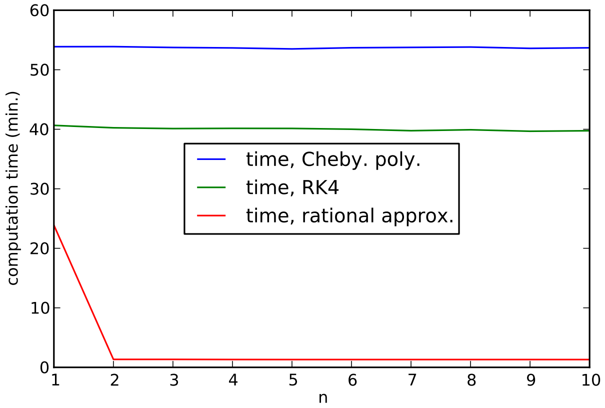


Table 2.2: Comparison of the accuracy and efficiency of applying, $e^{\tau\mathcal{L}}\mathbf{u}_0$ and $\tau = 1.5$, for system (2.6) and $\mathbf{u}_0$ in (2.29). The comparison uses RK4, Chebyshev polynomials, and the rational approximation (2.5); in the spatial discretization of all three comparisons, $12 \times 12 = 144$ elements and $16 \times 16 = 254$ Chebyshev quadrature nodes per element are used.

$e^{\tau\mathcal{L}}$, $\tau = 1.5$	L^{∞} error	time (min.)	pre-comp. (min.)
Rational approx., $M = 376$ terms	2.1×10^{-10}	4.39	103.1
RK4	7.0×10^{-10}	131.9	NA
Cheby. poly., degree 12	1.1×10^{-10}	150.5	NA

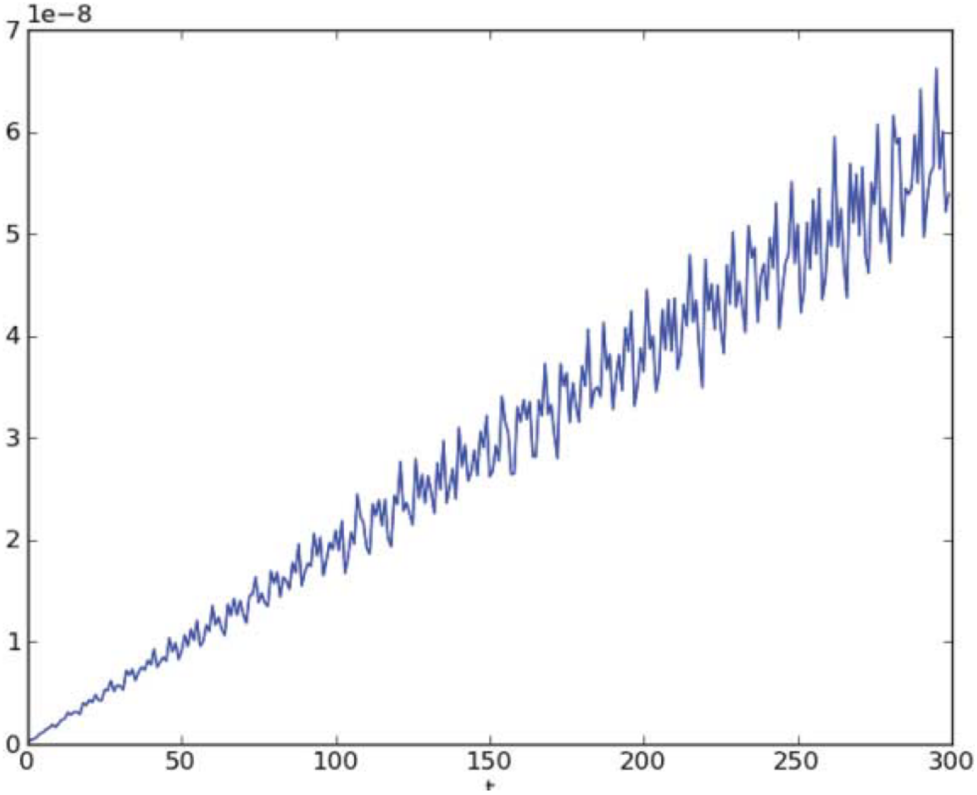


Figure 2.7: Plot of the L^∞ error, $\left\|\mathbf{u}_n - e^{n\tau L}\mathbf{u}_0\right\|_\infty$, versus the big time step $n\tau$, where $\tau = 1$ and $1 \le n \le 300$. Here $\mathbf{u}_n$ denotes the numerical approximation to $e^{n\tau L}\mathbf{u}_0$, as computed by the rational approximation (2.5) and the direct solver from Section 2.2.

since the matrix exponential is diagonalized in the Fourier domain, and can be rapidly applied via the Fast Fourier Transform (FFT). In particular,

$$\mathcal{L}\left(\mathbf{r}_{\mathbf{k}}^{l} e^{i\mathbf{k}\cdot\mathbf{x}}\right) = i\omega_{\mathbf{k}}^{l} \mathbf{r}_{\mathbf{k}}^{l} e^{i\mathbf{k}\cdot\mathbf{x}},$$

where $\mathbf{r}_{\mathbf{k}}^{l}$ are eigenvectors of the matrix

$$\begin{pmatrix} 0 & -f & igk_1 \\ -f & 0 & igk_2 \\ iHk_1 & iHk_2 & 0 \end{pmatrix},$$

and can be found in [56].

We first compare the accuracy and efficiency of applying $e^{n\tau L}\mathbf{u}_0$, for $\tau = 3$ and $n = 1, \ldots, 10$, against 4th order Runge-Kutta (RK4) and against using Chebyshev polynomials. In particular, the Chebyshev method uses the approximation

$$e^{\Delta t\mathcal{L}}\mathbf{u}_0 \approx J_0\left(i\right)\mathbf{u}_0 + 2\sum_{k=0}^{K}\left(i\right)^k J_k\left(-i\right) T_k\left(\Delta t\mathcal{L}\right)\mathbf{u}_0, \tag{2.28}$$

coupled with the standard recursion for applying $T_k\left(\Delta t L\right)$; we choose a polynomial degree of 12, which we find experimentally is a good compromise between the time step size Δt needed for a given accuracy, and the number of applications of $\mathcal{L}$. In all the time-stepping schemes, we use the same spectral element discretization and parameter values as described above. All the algorithms are implemented in Octave, including the direct solver described in Section 2.2.

2.4.1.1 First test case for the shallow water equations

We first consider the initial conditions

$$\begin{aligned} \eta\left(\mathbf{x}\right) &= \sin\left(6\pi x\right)\cos\left(4\pi y\right) - \frac{1}{5}\cos\left(4\pi x\right)\sin\left(2\pi y\right), \\ v_1\left(\mathbf{x}\right) &= \cos\left(6\pi x\right)\cos\left(4\pi y\right) - 4\sin\left(6\pi x\right)\sin\left(4\pi y\right), \\ v_2\left(\mathbf{x}\right) &= \cos\left(6\pi x\right)\cos\left(6\pi y\right). \end{aligned} \tag{2.29}$$

For these initial conditions, we use $6 \times 6 = 36$ elements of equal area, and $16 \times 16 = 256$ Chebyshev quadrature nodes for each element. To assess the accuracy of the method, the exponential $e^{n\tau\mathcal{L}}\mathbf{u}_0$

is applied in the Fourier domain. When applying the operator exponential using the rational approximation (2.5), we use $M = 376$ inverses and $\tau = 3$; this results in an L^{∞} error of 3.4×10^{-10} for a single (large) time step. For this choice of parameters in the spectral element discretization, the cost of applying the solution operator of (2.8)—i.e., forming the right hand side of (2.11), solving (2.11), and evaluating (2.10)—is about 4.5 times more expensive than the cost of applying the forward operator (2.7) directly.

For the three time-stepping methods, the L^{∞} errors in the approximation of $e^{n\tau\mathcal{L}}\mathbf{u}_0$, $n = 1, \ldots, 10$, are plotted in Figure 2.6, (a). Similarly, the total computation times (in minutes) of approximating $e^{n\tau\mathcal{L}}\mathbf{u}_0$, $n = 1, \ldots, 10$, are plotted in Figure 2.6, (b) (this includes the pre-computation time for representing the inverses). From Figure 2.6, (a), we see that the L^{∞} errors from all three methods remain less than 10^{-8} for $n = 1, \ldots, 10$. From Figure 2.6, (b), we see that the first time step for the rational approximation method is about half the cost of both RK4 and the Chebyshev polynomial method. However, subsequent time steps for the new method is about 40 times cheaper than both RK4 and the Chebyshev polynomial method (for about the same accuracy).

2.4.1.2 Second test case: doubling the spatial resolution

Next, we compute $e^{\tau\mathcal{L}}\mathbf{u}_0$, $\tau = 1.5$, with the initial conditions

$$\begin{aligned}
\eta\left(\mathbf{x}\right) &= \sin\left(12\pi x\right)\cos\left(8\pi y\right) - \frac{1}{5}\cos\left(8\pi x\right)\sin\left(4\pi y\right), \\
v_1\left(\mathbf{x}\right) &= \cos\left(12\pi x\right)\cos\left(8\pi y\right) - 4\sin\left(12\pi x\right)\sin\left(8\pi y\right), \\
v_2\left(\mathbf{x}\right) &= \cos\left(12\pi x\right)\cos\left(12\pi y\right). \qquad (2.30)
\end{aligned}$$

In particular, we double the bandlimit in each direction. In each of the time-stepping schemes, we use $12 \times 12 = 144$ elements of equal area, and $16 \times 16 = 256$ Chebyshev quadrature nodes for each element. We again use $M = 376$ inverses in (2.5).

We only examine the error and computation time for one big time step. For the rational approximation method, we present both the pre-computation time for obtaining data-sparse representations of the 376 inverses in (2.5), and the computation time for applying the approximation in

(2.5) (once the data-sparse representations are known). The results are summarized in Table 2.4.1. Since we only consider a single time step, the pre-computation time and application time are included separately. The main conclusion to draw from these results is that doubling the spatial resolution does not appreciably change the relative efficiency of the three time-stepping methods (once representations for the inverse operators in (2.5) are pre-computed).

2.4.1.3 Third test case: applying the operator exponential over a long time interval

Finally, we access the accuracy of the new method when repeatedly applying $e^{\tau\mathcal{L}}$, $\tau = 1$, in order to evolve the solution over longer time intervals. In this example, we use the initial conditions

$$\begin{aligned}
\eta(\mathbf{x}) &= \exp\left(-100\left((x-1/2)^2+(y-1/2)^2\right)\right), \\
v_1(\mathbf{x}) &= \cos(6\pi x)\cos(4\pi y) - 4\sin(6\pi x)\sin(4\pi y), \\
v_2(\mathbf{x}) &= \cos(6\pi x)\cos(6\pi y). \qquad (2.31)
\end{aligned}$$

Notice that these initial conditions cannot be expressed as a finite sum of eigenfunctions of $\mathcal{L}$. We use the same spatial discretization parameters as in Section 2.4.1.2.

In Figure 2.7, we show the L^{∞} error of the computed approximation $\mathbf{u}_n(\mathbf{x})$ to $\mathbf{u}(\mathbf{x}, n\tau)$, $n = 1, \ldots, 300$. As expected, the error increases linearly in the number of applications of the exponential. Notice that, due to the large step size of $\tau = 1$, the error accumulates slowly in time and the solution can be propagated with high accuracy over a large number of characteristic wavelengths.

2.4.2 Example 2

In our second example, we consider the wave propagation problem

$$u_{tt} = \kappa\Delta u, \quad \mathbf{x} \in [0,1]\times[0,1], \qquad (2.32)$$

where $\kappa(\mathbf{x}) > 0$ is a smooth function, the initial conditions $u(\mathbf{x}, 0)$ and $u_t(\mathbf{x}, 0)$ are prescribed, and periodic boundary conditions are used.

Table 2.3: Comparison of the accuracy and efficiency for the operator exponential, $e^{t\mathcal{L}}\mathbf{u}_0$ and $t = 1.5$, for system (2.13) and $\mathbf{u}_0$ in (2.33). The comparison uses RK4, Chebyshev polynomials, and the rational approximation (2.5); in the spatial discretization of all three comparisons, $12 \times 12 = 144$ elements and $16 \times 16 = 254$ Chebyshev quadrature nodes per element are used.

$e^{t\mathcal{L}}$, $t = 1.5$	L^∞ error	time (min.)	pre-comp. (min.)
Rational approx., $M = 376$ terms	1.6×10^{-9}	3.76	113.4
RK4	3.5×10^{-10}	63.9	NA
Cheby. poly., degree 12	3.5×10^{-8}	57.5	NA

Since the procedure and results are similar to those in Section 2.4.1, we simply test the efficiency and accuracy of this method over a single time step $\tau = 1.5$. In particular, we compare the accuracy and efficiency for one application $e^{\tau\mathcal{L}}\mathbf{u}_0$, $\tau = 1.5$, against 4th order Runge-Kutta (RK4) and against using Chebyshev polynomials. In our numerical experiments, we use the initial condition

$$u\left(x,y,0\right) = \sin\left(2\pi x\right)\sin\left(2\pi y\right) + \sin\left(4\pi x\right)\sin\left(4\pi y\right), \tag{2.33}$$

and $u_t\left(x,y,0\right) = 0$. We also use

$$\kappa\left(x,y\right) = \left(\frac{3+\sin\left(4\pi x\right)}{4}\right)^{1/2}\left(\frac{3+\sin\left(4\pi y\right)}{4}\right)^{1/2}.$$

Finally, in the spatial discretization, we use $12 \times 12 = 144$ elements with $16 \times 16 = 256$ points per element (for all three time-stepping methods), and $M = 376$ poles in (2.5). For these parameters, the time to apply the inverse of (2.14)—which involves forming the right hand side in (2.16), solving for v, and computing (2.15)—is about 5.2 times more expensive than directly applying the forward operator (2.13).

Unlike Section 2.4.1, the operator exponential is not diagonalized in the Fourier domain. To assess the accuracy, we use the Chebyshev polynomial method with a small enough step size to yield an estimated error of less than 10^{-10}. In particular, we verify that the L^∞ residual, $\left\|\mathbf{u}\left(\mathbf{x},t;\Delta t\right) - \mathbf{u}\left(\mathbf{x},t;\Delta t/2\right)\right\|_\infty$, using numerical approximations to $\mathbf{u}\left(\mathbf{x},t\right)$ computed with step sizes Δt and $\Delta t/2$ and the Chebyshev polynomial method, is less than 10^{-10}. We then use $\mathbf{u}\left(\mathbf{x},t;\Delta t/2\right)$

as a reference solution.

The results are summarized in Table 2.3. From this table, we see that the pre-computation time needed to represent the $M = 376$ solution operators in (2.5) is 93 minutes, and the computation time needed to apply the exponential is 3.7 minutes; the final accuracy in the L^∞ norm is given by 1.6×10^{-9}. For the Chebyshev polynomial method, 575 time steps of size $\Delta t \approx .0026$ are taken, for an overall time of 57 minutes; the final accuracy is given by 3.5×10^{-8}. Finally, for RK4, $7,500$ time steps of size $\Delta t = 1/5 \times 10^{-3}$ are taken, for an overall time of 63.9 minutes; the final accuracy is 3.5×10^{-10}.

2.5 Generalizations

The manuscript presents an efficient technique for explicitly computing a highly accurate approximation to the operator $\varphi(\tau\mathcal{L})$ for the case where $\mathcal{L}$ is a skew-Hermitian operator and where $\varphi(t) = e^t$, so that $\varphi(\tau\mathcal{L})$ is the time-evolution operator of the hyperbolic PDE $\partial u/\partial t = \mathcal{L}u$. The technique can be extended to more general functions φ. In particular, in using exponential integrators (cf. [46]), it is desirable to apply functions $\varphi_j(\tau\mathcal{L})$, where $\varphi_j(\cdot)$ are the so-called phi-functions. In Section 2.3, we presented (near) optimal rational approximations of the first few phi functions. An important property of these representations is that the same poles can be used to simultaneously apply all the phi-functions, and with a uniformly small error. In particular, linear combinations of the same $2M+1$ solutions $(\tau\mathcal{L} - \alpha_m)^{-1}\mathbf{u}_0$, $m = -M, \ldots, M$, can be used to apply $\varphi_j(\tau\mathcal{L})$ for $j = 1, 2, \ldots$. In a similar way, linear combinations of the same $2M+1$ solutions can be used to apply $e^{s\mathcal{L}}$ for $0 \leq s \leq \tau$.

In addition, where there is a priori knowledge of large spectral gaps—for example, when there is scale separation between fast and slow waves—the techniques in this paper, coupled with those in [52]), can be used to construct efficient rational approximations of e^{ix} which are (approximately) nonzero only where the spectrum of $\mathcal{L}$ is nonzero. Since suitably constructed rational approximations can capture functions with sharp transitions using a small number of poles (see [52]), this approach requires a potentially much smaller number of inverse applications.

Chapter 3

An accelerated Poisson solver based on multidomain spectral discretization

Note: The material in this chapter has appeared in press as: T. Babb, A. Gillman, S. Hao, and P.G. Martinsson. “An accelerated Poisson solver based on multidomain spectral discretization”. *BIT Numerical Mathematics*, **58**(4), 2018.

Abstract: This paper presents a numerical method for variable coefficient elliptic PDEs with mostly smooth solutions on two dimensional domains. The method works best for domains that can readily be mapped onto a rectangle, or a collection of nonoverlapping rectangles. The PDE is discretized via a multi-domain spectral collocation method of high local order (order 30 and higher have been tested and work well). Local mesh refinement results in highly accurate solutions even in the presence of local irregular behavior due to corner singularities, localized loads, etc. The system of linear equations attained upon discretization is solved using a direct (as opposed to iterative) solver with $O(N^{1.5})$ complexity for the factorization stage and $O(N \log N)$ complexity for the solve. The scheme is ideally suited for executing the elliptic solve required when parabolic problems are discretized via time-implicit techniques. In situations where the geometry remains unchanged between time-steps, very fast execution speeds are obtained since the solution operator for each implicit solve can be pre-computed.

3.1 Introduction

This manuscript describes a direct solver for elliptic PDEs such as, e.g.,

$$\begin{cases} [Au](\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma, \end{cases} \tag{3.1}$$

where A is a variable coefficient elliptic differential operator

$$[Au](\boldsymbol{x}) = -c_{11}(\boldsymbol{x})[\partial_1^2 u](\boldsymbol{x}) - 2c_{12}(\boldsymbol{x})[\partial_1\partial_2 u](\boldsymbol{x}) - c_{22}(\boldsymbol{x})[\partial_2^2 u](\boldsymbol{x}) + c_1(\boldsymbol{x})[\partial_1 u](\boldsymbol{x}) + c_2(\boldsymbol{x})[\partial_2 u](\boldsymbol{x}) + c(\boldsymbol{x})\,u(\boldsymbol{x}), \tag{3.2}$$

where Ω is a rectangular domain in $\mathbb{R}^2$ with boundary $\Gamma = \partial\Omega$, where all coefficient functions (c, c_i, c_{ij}) are smooth, and where f and g are given functions. The generalization to domains that are either unions of rectangles, or can via local parameterizations be mapped to a union of rectangles is relatively straight-forward [29, Sec. 6.4]. The technique is specifically developed to accelerate implicit time stepping techniques for parabolic PDEs such as, e.g., the heat equation

$$\begin{cases} \Delta u(\boldsymbol{x},t) = \dfrac{\partial u}{\partial t}(\boldsymbol{x},t), & \boldsymbol{x} \in \Omega,\ t > 0, \\ u(\boldsymbol{x},t) = f(\boldsymbol{x},t), & \boldsymbol{x} \in \Gamma,\ t > 0, \\ u(\boldsymbol{x},0) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega. \end{cases} \tag{3.3}$$

When (3.3) is discretized using an implicit time-stepping scheme (e.g. backwards Euler, Crank-Nicolson or the Transpose Method of Lines [7]), one is required to solve for each time-step an equation of the form (3.1), see Section 3.7.6. With the ability to combine very high order discretizations with a highly efficient means of time-stepping parabolic equations, we believe that the proposed method will be particularly well suited for numerically solving the Navier-Stokes equation at low Reynolds numbers.

The proposed solver is direct and builds an approximation to the solution operator of (3.1) via a hierarchical divide-and-conquer approach. It is conceptually related to classical nested dissection and multifrontal methods [8, 9, 14], but provides tight integration between the direct solver and the discretization procedure. Locally, the scheme relies on high order spectral discretizations, and collocation of the differential operator. We observe that while classical nested dissection and multifrontal solvers slow down dramatically as the discretization order is increased [16, Table 3], the proposed method retains high efficiency regardless of the discretization order. The method is an evolution of the scheme described in [29, 28], and later refined in [15, 16, 23]. One novelty of the present work is that it describes how problems with body loads can be handled efficiently (the previous papers [29, 15, 16, 23] consider the case where $g = 0$ in (3.1)). A second novelty is that local mesh refinement is introduced to enable the method to accurately solve problems involving concentrated loads, singularities at re-entrant corners, and other phenomena that lead to localized loss of regularity in the solution. (In contrast, the previous papers [29, 15, 16, 23] restrict attention to uniform grids.)

The principal advantage of the proposed solver, compared to commonly used solvers for (3.1), is that it is *direct* (as opposed to *iterative*), which makes it particularly well suited for problems for which efficient pre-conditioners are difficult to find, such as, e.g., problems with oscillatory solutions. The cost to build the solution operator is in the most basic version of the scheme $O(N^{3/2})$, where N is the number of discretization points. However, the practical efficiency of the solver is very high and the superlinear scaling is hardly visible until $N > 10^7$. When the number of discretization points is higher than 10^7, the scheme can be modified to attain linear complexity by implementing techniques analogous to those described in [15]. Once the solution operator has been built, the time required to apply it to execute a solve given a boundary condition and a body load is either $O(N \log N)$ for the basic scheme, or $O(N)$ for the accelerated scheme, with a small scaling constant in either case. In Section 3.7, we demonstrate that even when $N = 10^6$, the time for solving (3.1) with a precomputed solution operator is approximately one second on a standard office laptop.

The discretization scheme we use is related to earlier work on spectral collocation methods on composite ("multi-domain") grids, such as, e.g., [27, 41], and in particular Pfeiffer *et al* [33]. The differences and similarities between the various techniques is discussed in detail in [29]. Our procedure is also conceptually related to so-called "reduction to the interface" methods, see [26] and the references therein. Such “interface” methods also use local solution operators defined on boundaries but typically rely on variational formulations of the PDE, rather than the collocation techniques that we employ.

The manuscript is organized as follows: Section 3.2 provides a high level description of the proposed method. Sections 3.3 and 3.4 describe the local discretization scheme. Section 3.5 describes the nested dissection type solver used to solve the system of linear equations resulting from the discretization. Section 3.6 describes how local mesh refinement can be introduced to the scheme. Section 3.7 provides results from numerical experiments that establish the efficiency of the proposed method.

3.2 Overview of algorithm

The proposed method is based on a hierarchical subdivision of the computational domain, as illustrated in Figure 3.1 for the case of $\Omega = [0,1]^2$. In the uniform mesh version of the solver, the tree of boxes is built by recursively splitting the original box in halves. The splitting continues until each box is small enough that the solution, and its first and second derivatives, can accurately be resolved on a local tensor product grid of $p \times p$ Chebyshev nodes (where, say, $p = 10$ or $p = 20$).

Once the tree of boxes has been constructed, the actual solver consists of two stages. The first, or “build”, stage consists of a single upwards pass through the tree of boxes, starting with the leaves and going up to larger boxes. On each leaf, we place a local $p \times p$ tensor product grid of Chebyshev nodes, and then discretize the restriction of (3.1) via a basic collocation scheme, as in [37]. By performing dense linear algebraic operations on matrices of size at most $p^2 \times p^2$, we form for each leaf a local solution operator and an approximation to the local Dirichlet-to-Neumann (DtN) operator, as described in Section 3.3. The build stage then continues with an upwards pass through the tree (going from smaller boxes to larger) where for each parent box, we construct approximations to its local solution operator and its local DtN operator by “merging” the corresponding operators for its children, cf. Section 3.4. The end result of the “build stage” is a hierarchical representation of the overall solution operator for (3.1). Once this solution operator is available, the “solve stage” takes as input a given boundary data f and a body load g, and constructs an approximation to the solution u valid throughout the domain via two passes through the tree: first an upwards pass (going from smaller boxes to larger) where “particular solutions” that satisfy the inhomogeneous equation are built, and then a downwards pass where the boundary conditions are corrected.

The global grid of collocation points used in the upwards and downwards passes is obtained by placing on the edge of each leaf a set of q Gaussian interpolation nodes (a.k.a. Legendre nodes). Observe that this parameter q is in principle distinct from the local parameter p which specifies the order of the local Chebyshev grids used to construct the solution operators on the leaves. However, we typically choose $p = q + 1$ or $p = q + 2$.

3.3 Leaf computation

In this section, we describe how to numerically build the various linear operators (represented as dense matrices) needed for a given leaf Ω_τ in the hierarchical tree. To be precise, let u be the solution to the local equation

$$\begin{cases} [Au](\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega_\tau, \\ u(\boldsymbol{x}) = d(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma_\tau, \end{cases} \tag{3.4}$$

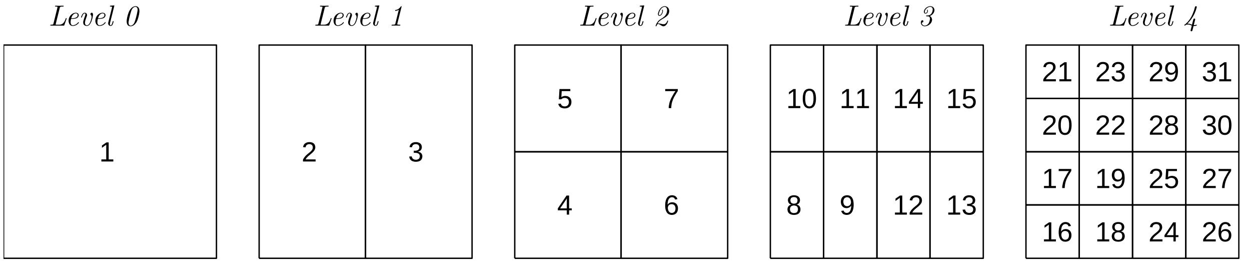


Figure 3.1: The square domain Ω is split into 4×4 leaf boxes. These are then gathered into a binary tree of successively larger boxes as described in Section 3.2. One possible enumeration of the boxes in the tree is shown, but note that the only restriction is that if box τ is the parent of box σ, then $\tau < \sigma$.

for some given (local) Dirichlet data d. We then build approximations to two linear operators that both take d and g as their inputs. The first operator outputs the local solution u on Ω_τ and the second outputs the boundary fluxes of u on Γ_τ.

3.3.1 Notation

We use two sets of interpolation nodes on the domain Ω_τ. First, let $\{\boldsymbol{y}_j\}_{j=1}^{4q}$ denote the nodes obtained by placing q Gaussian nodes on each of the four sides of Ω_τ. Next, let $\{\boldsymbol{x}_i\}_{i=1}^{p^2}$ denote the nodes in a $p \times p$ Chebyshev grid on Ω_τ. We partition the index vector for the nodes in the Chebyshev grid as

$$\{1, 2, \dots, p^2\} = I_{\text{ce}} \cup I_{\text{ci}}$$

so that I_{ce} holds the (<u>C</u>hebyshev) <u>e</u>xterior nodes and I_{ci} holds the (<u>C</u>hebyshev) <u>i</u>nterior nodes. Let u_{c}, u_{ci}, u_{ce}, and u_{ge} denote vectors holding approximations to the values of the solution u at the interpolation nodes:

$$\mathsf{u}_{\text{c}} \approx \{u(\boldsymbol{x}_i)\}_{i=1}^{p^2}, \qquad \mathsf{u}_{\text{ci}} \approx \{u(\boldsymbol{x}_i)\}_{i \in I_{\text{ci}}},$$

$$\mathsf{u}_{\text{ce}} \approx \{u(\boldsymbol{x}_i)\}_{i \in I_{\text{ce}}}, \qquad \mathsf{u}_{\text{ge}} \approx \{u(\boldsymbol{y}_j)\}_{j=1}^{4q}.$$

Let $\mathsf{v}_{\text{ge}} \in \mathbb{R}^{4q}$ denote a vector holding boundary fluxes of u on the Gaussian grid, so that

$$\mathsf{v}_{\text{ge}}(j) \approx [\partial_1 u](\boldsymbol{y}_{hj}) \qquad \text{when } \boldsymbol{y}_j \text{ lies on a vertical boundary,}$$

$$\mathsf{v}_{\text{ge}}(j) \approx [\partial_2 u](\boldsymbol{y}_{hj}) \qquad \text{when } \boldsymbol{y}_j \text{ lies on a horizontal boundary.}$$

The sign convention for boundary fluxes means that a positive flux sometimes represents flow into the box and sometimes out of the box. Finally, let d_{ge} and g_{ci} denote tabulations of the boundary data and the body load,

$$\mathsf{d}_{\text{ge}} = \{d(\boldsymbol{y}_j)\}_{j=1}^{4q}, \qquad \mathsf{g}_{\text{ci}} = \{g(\boldsymbol{x}_i)\}_{i \in I_{\text{ci}}}.$$

Our objective is now to build the matrices that map $\{\mathsf{d}_{\text{ge}}, \mathsf{g}_{\text{ci}}\}$ to v_{ge} and u_{c}.

3.3.2 Discretization on the Cheyshev grid

In order to execute the local solve on Ω_τ of (3.4), we use a classical spectral collocation technique, as described, e.g., in [37]. To this end, let $\mathsf{D}^{(1)}$ and $\mathsf{D}^{(2)}$ denote the $p^2 \times p^2$ spectral differentiation matrices on the $p \times p$ Chebyshev grid. (In other words, for any function u that is a tensor product of polynomials of degree at most $p-1$, the differentiation matrix *exactly* maps a vector of collocated function values to the vector of collocated values of its derivative.) Further, let A denote the matrix

$$\mathsf{A} = -\mathsf{C}_{11}(\mathsf{D}^{(1)})^2 - 2\mathsf{C}_{12}\mathsf{D}^{(1)}\mathsf{D}^{(2)} - \mathsf{C}_{22}(\mathsf{D}^{(2)})^2 + \mathsf{C}_1\mathsf{D}^{(1)} + \mathsf{C}_2\mathsf{D}^{(2)} + \mathsf{C},$$

where C_{ij} are diagonal matrices with entries $\{c_{ij}(\boldsymbol{x}_k)\}_{k=1}^{p^2}$, and C_i and C are defined analogously. Next, partition the matrix A to separate interior and exterior nodes via

$$\mathsf{A}_{\rm ci,ci} = \mathsf{A}(I_{\rm ci}, I_{\rm ci}), \qquad \text{and} \qquad \mathsf{A}_{\rm ci,ce} = \mathsf{A}(I_{\rm ci}, I_{\rm ce}).$$

Collocating (3.4) at the interior nodes then results in the discretized equation

$$\mathsf{A}_{\rm ci,ci}\,\mathsf{u}_{\rm ci} + \mathsf{A}_{\rm ci,ce}\,\mathsf{d}_{\rm ce} = \mathsf{g}_{\rm ci}, \tag{3.5}$$

where $\mathsf{d}_{\rm ce} = \{d(\boldsymbol{x}_i)\}_{i\in I_{\rm ce}}$ encodes the local Dirichlet data d.

3.3.3 Solving on the Chebyshev grid

While solving (3.5) gives the solution at the interior Chebychev nodes, it does not give a map to the boundary fluxes $\mathsf{v}_{\rm ge}$ that we seek. These are found by following the classic approach of writing the solution as the superposition of the homogeneous and particular solutions. Specifically, the solution to (3.4) is split as

$$u = w + \phi$$

where w is a *particular solution*

$$\begin{cases} Aw(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega_\tau, \\ w(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Gamma_\tau, \end{cases} \tag{3.6}$$

and where ϕ is a *homogeneous solution*

$$\begin{cases} A\phi(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Omega_\tau, \\ \phi(\boldsymbol{x}) = d(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma_\tau. \end{cases} \tag{3.7}$$

Discretizing (3.6) on the Chebyshev grid, and collocating at the internal nodes, we get the equation

$$\mathbf{A}_{\mathrm{ci,ce}}\mathbf{w}_{\mathrm{ce}} + \mathbf{A}_{\mathrm{ci,ci}}\mathbf{w}_{\mathrm{ci}} = \mathbf{g}_{\mathrm{ci}}.$$

Observing that $\mathbf{w}_{\mathrm{ce}} = 0$, the particular solution is given by

$$\mathbf{w}_{\mathrm{c}} = \begin{bmatrix} \mathbf{w}_{\mathrm{ce}} \\ \mathbf{w}_{\mathrm{ci}} \end{bmatrix} = \mathbf{F}_{\mathrm{c,ci}}\mathbf{g}_{\mathrm{ci}}, \qquad \text{where} \qquad \mathbf{F}_{\mathrm{c,ci}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{A}_{\mathrm{ci,ci}}^{-1} \end{bmatrix}. \tag{3.8}$$

Analogously, the discretization of (3.7) on the Chebyshev grid yields

$$\mathbf{A}_{\mathrm{ci,ce}}\boldsymbol{\phi}_{\mathrm{ce}} + \mathbf{A}_{\mathrm{ci,ci}}\boldsymbol{\phi}_{\mathrm{ci}} = \mathbf{0}.$$

Since $\boldsymbol{\phi}_{\mathrm{ce}} = \mathbf{d}_{\mathrm{ce}}$, the homogeneous solution is given by

$$\boldsymbol{\phi}_{\mathrm{c}} = \begin{bmatrix} \boldsymbol{\phi}_{\mathrm{ce}} \\ \boldsymbol{\phi}_{\mathrm{ci}} \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ -\mathbf{A}_{\mathrm{ci,ci}}^{-1}\mathbf{A}_{\mathrm{ci,ce}} \end{bmatrix} \mathbf{d}_{\mathrm{ce}}. \tag{3.9}$$

3.3.4 Interpolation and differentiation

Section 3.3.3 describes how to locally solve the BVP (3.4) on the Chebyshev grid via the superposition of the homogeneous and particular solutions. This computation assumes that the local Dirichlet data d is given on the *Chebyshev* exterior nodes. In reality, this data will be provided on the Gaussian nodes, and we therefore need to introduce an interpolation operator that moves data between the different grids. To be precise, let $\mathbf{L}_{\mathrm{ce,ge}}$ denote a matrix of size $4(p-1) \times 4q$ that maps a given data vector $\mathbf{d}_{\mathrm{ge}}$ to a different vector

$$\begin{array}{ccccc} \mathbf{d}_{\mathrm{ce}} & = & \mathbf{L}_{\mathrm{ce,ge}} & \mathbf{d}_{\mathrm{ge}} \\ 4(p-1) \times 1 & & 4(p-1) \times 4q & 4q \times 1 \end{array} \tag{3.10}$$

as follows: An entry of $\mathbf{d}_{\mathrm{ce}}$ corresponding to an interior node is defined via a standard interpolation from the Gaussian to the Chebyshev nodes on the local edge. An entry of $\mathbf{d}_{\mathrm{ce}}$ corresponding to a corner node is defined as the average value of the two extrapolated values from the Gaussian nodes on the two edges connecting to the corner. (Observe that except for the four rows corresponding to the corner nodes, $\mathsf{L}_{\rm ce,ge}$ is a 4×4 block diagonal matrix.)

Combining (3.8), (3.9), and (3.10), the solution to (3.4) on the Chebyshev grid is

$$\mathsf{u}_{\rm c} = \mathsf{w}_{\rm c} + \phi_{\rm c} = \mathsf{F}_{\rm c,ci}\,\mathsf{g}_{\rm ci} + \mathsf{S}_{\rm c,ge}\,\mathsf{d}_{\rm ge}, \quad \text{where} \quad \mathsf{S}_{\rm c,ge} := \begin{bmatrix} \mathsf{I}_{\rm ce,ce} \\ -\mathsf{A}_{\rm ci,ci}^{-1}\mathsf{A}_{\rm ci,ce} \end{bmatrix} \mathsf{L}_{\rm ce,ge}. \tag{3.11}$$

All that remains is to determine the vector $\mathsf{v}_{\rm ge}$ of boundary fluxes on the Gaussian nodes. To this end, let us define a combined interpolation and differentiation matrix $\mathsf{D}_{\rm ge,c}$ of size $4q \times p^2$ via

$$\mathsf{D}_{\rm ge,c} = \begin{bmatrix} \mathsf{L}_{\rm loc}\,\mathsf{D}_2(I_{\rm s},:) \\ \mathsf{L}_{\rm loc}\,\mathsf{D}_1(I_{\rm e},:) \\ \mathsf{L}_{\rm loc}\,\mathsf{D}_2(I_{\rm n},:) \\ \mathsf{L}_{\rm loc}\,\mathsf{D}_1(I_{\rm w},:) \end{bmatrix},$$

where $\mathsf{L}_{\rm loc}$ is a $q \times p$ interpolation matrix from a set of p Chebyshev nodes to a set of q Gaussian nodes, and where $I_{\rm s}$, $I_{\rm e}$, $I_{\rm n}$, $I_{\rm w}$ are vectors of length p with entries corresponding to the points on the south, east, north, and west sides of the exterior nodes in the Chebyshev grid. By differentiating the local solution on the Chebyshev grid defined by (3.11), the boundary fluxes $\mathsf{v}_{\rm ge}$ are given by

$$\mathsf{v}_{\rm ge} = \mathsf{H}_{\rm ge,ci}\,\mathsf{g}_{\rm ci} + \mathsf{T}_{\rm ge,ge}\,\mathsf{d}_{\rm ge}, \tag{3.12}$$

where

$$\mathsf{H}_{\rm ge,ci} = \mathsf{D}_{\rm ge,c}\,\mathsf{F}_{\rm c,ci} \qquad \text{and} \qquad \mathsf{T}_{\rm ge,ge} = \mathsf{D}_{\rm ge,c}\,\mathsf{S}_{\rm c,ge}. \tag{3.13}$$

3.4 Merging two leaves

Consider a rectangular box τ consisting of two leaf boxes α and β, and suppose that all local operators for α and β defined in Section 3.3 have been computed. Our objective is now to construct the Dirichlet-to-Neumann operator for τ from the local operators for its children. In this operation, only sets of Gaussian nodes on the boundaries will take part, cf. Figure 3.2. We group these nodes into three sets, indexed by vectors J_1, J_2, and J_3, defined as follows:

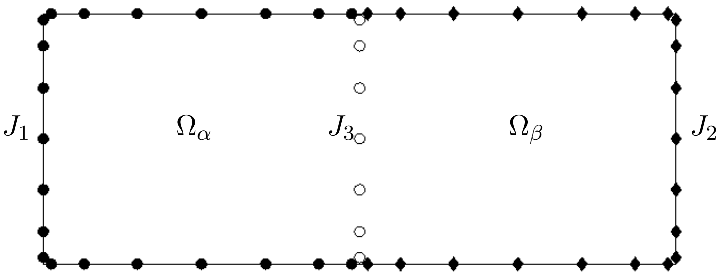


Figure 3.2: Notation for the merge operation described in Section 3.4. Given two leaf boxes Ω_α and Ω_β, their union is denoted $\Omega_\tau = \Omega_\alpha \cup \Omega_\beta$. The sets J_1 (black circles) and J_2 (black diamonds) form the exterior nodes, while J_3 (white circles) consists of the interior nodes.

J_1	Edge nodes of box α that are not shared with box β.
J_2	Edge nodes of box β that are not shared with box α.
J_3	Edge nodes that line the interior edge shared by α and β.

We also define

$$J_{\text{ge}}^{\tau} = J_1 \cup J_2 \qquad \text{and} \qquad J_{\text{gi}}^{\tau} = J_3$$

as the exterior and interior nodes for the parent box τ. Finally, we let $\mathbf{h}^\alpha, \mathbf{h}^\beta \in \mathbb{R}^{4q}$ denote two vectors that hold the boundary fluxes for the two local particular solutions w^α and w^β, cf. (3.12),

$$\mathbf{h}_{\text{ge}}^{\alpha} = \mathbf{H}_{\text{ge,ci}}^{\alpha}\,\mathbf{g}_{\text{ci}}^{\alpha}, \qquad \text{and} \qquad \mathbf{h}_{\text{ge}}^{\beta} = \mathbf{H}_{\text{ge,ci}}^{\beta}\,\mathbf{g}_{\text{ci}}^{\beta}. \tag{3.14}$$

Then the equilibrium equations for each of the two leaves can be written

$$\mathbf{v}_{\text{ge}}^{\alpha} = \mathbf{T}_{\text{ge,ge}}^{\alpha}\,\mathbf{u}_{\text{ge}}^{\alpha} + \mathbf{h}_{\text{ge}}^{\alpha}, \qquad \text{and} \qquad \mathbf{v}_{\text{ge}}^{\beta} = \mathbf{T}_{\text{ge,ge}}^{\beta}\,\mathbf{u}_{\text{ge}}^{\beta} + \mathbf{h}_{\text{ge}}^{\beta}. \tag{3.15}$$

Now partition the two equations in (3.15) using the notation shown in Figure 3.2:

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}_{1,1}^{\alpha} & \mathbf{T}_{1,3}^{\alpha} \\ \mathbf{T}_{3,1}^{\alpha} & \mathbf{T}_{3,3}^{\alpha} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_3 \end{bmatrix} + \begin{bmatrix} \mathbf{h}_1^{\alpha} \\ \mathbf{h}_3^{\alpha} \end{bmatrix}, \tag{3.16}$$

$$\begin{bmatrix} \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}_{2,2}^{\beta} & \mathbf{T}_{2,3}^{\beta} \\ \mathbf{T}_{3,2}^{\beta} & \mathbf{T}_{3,3}^{\beta} \end{bmatrix} \begin{bmatrix} \mathbf{u}_2 \\ \mathbf{u}_3 \end{bmatrix} + \begin{bmatrix} \mathbf{h}_2^{\beta} \\ \mathbf{h}_3^{\beta} \end{bmatrix}. \tag{3.17}$$

(The subscript "ge" is suppressed in (3.16) and (3.17) since all nodes involved are Gaussian exterior nodes.) Combine the two equations for $\mathbf{v}_3$ in (3.16) and (3.17) to obtain the equation

$$\mathbf{T}^{\alpha}_{3,1}\,\mathbf{u}_1 + \mathbf{T}^{\alpha}_{3,3}\,\mathbf{u}_3 + \mathbf{h}^{\alpha}_3 = \mathbf{T}^{\beta}_{3,2}\,\mathbf{u}_2 + \mathbf{T}^{\beta}_{3,3}\,\mathbf{u}_3 + \mathbf{h}^{\beta}_3.$$

This gives

$$\mathbf{u}_3 = \big(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\big)^{-1}\big(\mathbf{T}^{\beta}_{3,2}\mathbf{u}_2 - \mathbf{T}^{\alpha}_{3,1}\mathbf{u}_1 + \mathbf{h}^{\beta}_3 - \mathbf{h}^{\alpha}_3\big) \tag{3.18}$$

Using the relation (3.18) in combination with (3.16), we find that

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix} = \left(\begin{bmatrix} \mathbf{T}^{\alpha}_{1,1} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \big(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\big)^{-1}\big[-\mathbf{T}^{\alpha}_{3,1} \mid \mathbf{T}^{\beta}_{3,2}\big]. \right) \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix} +$$
$$\begin{bmatrix} \mathbf{h}^{\alpha}_1 \\ \mathbf{h}^{\beta}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \big(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\big)^{-1}\big(\mathbf{h}^{\beta}_3 - \mathbf{h}^{\alpha}_3\big).$$

We now define the operators

$$\begin{aligned}
\mathbf{X}^{\tau}_{\mathrm{gi,gi}} &= \big(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\big)^{-1}, \\
\mathbf{S}^{\tau}_{\mathrm{gi,ge}} &= \big(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\big)^{-1}\big[-\mathbf{T}^{\alpha}_{3,1} \mid \mathbf{T}^{\beta}_{3,2}\big] = \mathbf{X}^{\tau}_{\mathrm{gi,gi}}\big[-\mathbf{T}^{\alpha}_{3,1} \mid \mathbf{T}^{\beta}_{3,2}\big], \\
\mathbf{T}^{\tau}_{\mathrm{ge,ge}} &= \begin{bmatrix} \mathbf{T}^{\alpha}_{1,1} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \big(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\big)^{-1}\big[-\mathbf{T}^{\alpha}_{3,1} \mid \mathbf{T}^{\beta}_{3,2}\big] \\
&= \begin{bmatrix} \mathbf{T}^{\alpha}_{1,1} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \mathbf{S}^{\tau}_{\mathrm{gi,ge}}.
\end{aligned}$$

The approximate solution on the shared edge $\mathbf{u}^{\tau}_{\mathrm{gi}}$ can be constructed via an upward pass to compute the approximate boundary flux by

$$\mathbf{h}^{\tau}_{\mathrm{ge}} = \begin{bmatrix} \mathbf{h}^{\alpha}_1 \\ \mathbf{h}^{\beta}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \mathbf{w}^{\tau}_{\mathrm{gi}}, \tag{3.19}$$

where $\mathbf{w}^{\tau}_{\mathrm{gi}} = \mathbf{X}^{\tau}_{\mathrm{gi,gi}}\big(\mathbf{h}^{\beta}_3 - \mathbf{h}^{\alpha}_3\big)$, followed by a downward pass

$$\mathbf{u}^{\tau}_{\mathrm{gi}} = \mathbf{S}_{\mathrm{gi,ge}}\mathbf{u}^{\tau}_{\mathrm{ge}} + \mathbf{w}^{\tau}_{\mathrm{gi}}.$$

Remark 2 (Physical interpretation of merge) The quantities $\mathsf{w}_{\rm gi}^{\tau}$ and $\mathsf{h}_{\rm ge}^{\tau}$ have a simple physical meaning. The vector $\mathsf{w}_{\rm gi}^{\tau}$ introduced above is simply a tabulation of the particular solution w^{τ} associated with τ on the interior boundary Γ_3, and $\mathsf{h}_{\rm ge}^{\tau}$ is the normal derivative of w^{τ}. To be precise, w^{τ} is the solution to the inhomogeneous problem, cf. (3.6)

$$\begin{cases} Aw^{\tau}(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega_{\tau}, \\ w^{\tau}(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Gamma_{\tau}. \end{cases} \tag{3.20}$$

We can re-derive the formula for $w|_{\Gamma_3}$ using the original mathematical operators as follows: First observe that for $\boldsymbol{x} \in \Omega^{\alpha}$, we have $A(w^{\tau} - w^{\alpha}) = g - g = 0$, so the DtN operator T^{α} applies to the function $w^{\tau} - w^{\alpha}$:

$$T_{31}^{\alpha}(w_1^{\tau} - w_1^{\alpha}) + T_{33}^{\alpha}(w_3^{\tau} - w_3^{\alpha}) = (\partial_n w^{\tau})|_3 - (\partial_n w^{\alpha})|_3$$

Use that $w_1^{\tau} = w_1^{\alpha} = w_3^{\alpha} = 0$, and that $(\partial_n w^{\alpha})|_3 = h_3^{\alpha}$ to get

$$T_{33}^{\alpha} w_3^{\tau} = (\partial_n w^{\tau})|_3 - h_3^{\alpha}. \tag{3.21}$$

Analogously, we get

$$T_{33}^{\beta} w_3^{\tau} = (\partial_n w^{\tau})|_3 - h_3^{\beta}. \tag{3.22}$$

Combine (3.21) and (3.22) to eliminate $(\partial_n w^{\tau})|_3$ and obtain

$$\big(T_{33}^{\alpha} - T_{33}^{\beta}\big) w_3^{\tau} = -h_3^{\alpha} + h_3^{\beta}.$$

Observe that in effect, we can write the particular solution w^{τ} as

$$w^{\tau}(\boldsymbol{x}) = \begin{cases} w^{\alpha}(\boldsymbol{x}) + \hat{w}^{\tau}(\boldsymbol{x}) & \boldsymbol{x} \in \Omega^{\alpha}, \\ w^{\beta}(\boldsymbol{x}) + \hat{w}^{\tau}(\boldsymbol{x}) & \boldsymbol{x} \in \Omega^{\beta}, \end{cases}$$

The function w^{τ} must of course be smooth across Γ_3, so the function $\hat{w}^{\tau}$ must have a jump that exactly offsets the discrepancy in the derivatives of w^{α} and w^{β}. This jump is precisely of size $h^{\alpha} - h^{\beta}$.

3.5 The full solver for a uniform grid

3.5.1 Notation

Suppose that we are given a rectangular domain Ω, which has hierarchically been split into a binary tree of successively smaller patches, as described in Section 3.2. We then define two sets of interpolation nodes. First, $\{\boldsymbol{x}_i\}_{i=1}^{M}$ denotes the set of nodes obtained by placing a $p \times p$ tensor product grid of Chebyshev nodes on each leaf in the tree. For a leaf τ, let I_{c}^{τ} denote an index vector pointing to the nodes in $\{\boldsymbol{x}_i\}_{i=1}^{M}$ that lie on leaf τ. Thus the index vector for the set of nodes in τ can be partitioned into exterior and interior nodes as follows

$$I_{\mathrm{c}}^{\tau} = I_{\mathrm{ce}}^{\tau} \cup I_{\mathrm{ci}}^{\tau}.$$

The second set of interpolation nodes $\{\boldsymbol{y}_j\}_{j=1}^{N}$ is obtained by placing a set of q Gaussian ("Legendre") interpolation nodes on the edge of each leaf. For a node τ in the tree (either a leaf or a parent), let I_{ge}^{τ} denote an index vector that marks all Gaussian nodes that lie on the boundary of Ω_{τ}. For a parent node τ, let I_{gi}^{τ} denote the Gaussian nodes that are interior to τ, but exterior to its two children (as in Section 3.4).

Once the collocation points have been set up, we introduce a vector $\mathbf{u} \in \mathbb{R}^{M}$ holding approximations to the values of the potential u on the Gaussian collocation points,

$$\mathbf{u}(j) \approx u(\boldsymbol{y}_j), \qquad j = 1,\, 2,\, 3,\, \ldots,\, M.$$

We refer to subsets of this vector using the short-hand

$$\mathbf{u}_{\mathrm{ge}}^{\tau} = \mathbf{u}(I_{\mathrm{ge}}^{\tau}), \qquad \text{and} \qquad \mathbf{u}_{\mathrm{gi}}^{\tau} = \mathbf{u}(I_{\mathrm{gi}}^{\tau})$$

for the exterior and interior nodes respectively. At the very end of the algorithm, approximations to u on the local Chebyshev tensor product grids are constructed. For a leaf node τ, let the vectors $\mathbf{u}_{\mathrm{c}}^{\tau}$, $\mathbf{u}_{\mathrm{ce}}^{\tau}$, and $\mathbf{u}_{\mathrm{ci}}^{\tau}$ denote the vectors holding approximations to the potential on sets of collocation points in the Chebyshev grid marked by I_{c}^{τ}, I_{ce}^{τ}, and I_{ci}^{τ}, respectively. Observe that these vectors are *not* subvectors of $\mathbf{u}$.

Before proceeding to the description of the algorithm, we introduce two sets of auxiliary vectors. First, for any parent node τ, let the vector $\mathbf{w}_{\text{gi}}^{\tau}$ denote the computed values of the local particular solution w^{τ} that solves (3.6) on Ω_{τ}, as tabulated on the interior line marked by I_{gi}^{τ}. Also, define $\mathbf{h}^{\tau}$ as the approximate boundary fluxes of w^{τ} as defined by (3.14) for a leaf and by (3.19) for a parent.

3.5.2 The build stage

Once the domain is partitioned into a hierarchical tree, we execute a "build stage" in which the following matrices are constructed for each box τ:

$\mathbf{S}^{\tau}$ For a box τ, the solution operator that maps Dirichlet data ψ on $\partial\Omega_{\tau}$ to values of u at the interior nodes. In other words, $\mathbf{u}_{\text{c}}^{\tau} = \mathbf{S}_{\text{c,ge}}^{\tau}\boldsymbol{\psi}_{\text{ge}}^{\tau}$ on a leaf or $\mathbf{u}_{\text{gi}}^{\tau} = \mathbf{S}_{\text{gi,ge}}^{\tau}\boldsymbol{\psi}_{\text{ge}}^{\tau}$ on a parent box.

$\mathbf{T}^{\tau}$ For a box τ, the matrix that maps Dirichlet data ψ on $\partial\Omega_{\tau}$ to the flux v on the boundary. In other words, $\mathbf{v}_{\text{ge}}^{\tau} = \mathbf{T}_{\text{ge,ge}}^{\tau}\boldsymbol{\psi}_{\text{ge}}$.

$\mathbf{F}^{\tau}$ For a leaf box, the matrix that maps the body load to the particular solution on the interior of the leaf assuming the Dirichlet data is zero on the boundary. In other words $\mathbf{w}_{\text{c}}^{\tau} = \mathbf{F}_{\text{c,ci}}^{\tau}\mathbf{g}_{\text{ci}}$.

$\mathbf{H}^{\tau}$ For a leaf box, the matrix that maps the body load to the flux on the boundary of the leaf. In other words $\mathbf{h}_{\text{ge}}^{\tau} = \mathbf{H}_{\text{ge,ci}}^{\tau}\mathbf{g}_{\text{ci}}^{\tau}$.

$\mathbf{X}^{\tau}$ For a parent box τ with children α and β, $\mathbf{X}^{\tau}$ maps the fluxes of the particular solution for the children on the interior of a parent to the particular solution on the interior nodes. In other words $\mathbf{w}_{\text{gi}} = \mathbf{X}_{\text{gi,gi}}^{\tau}(\mathbf{h}_3^{\beta} - \mathbf{h}_3^{\alpha})$.

The build stage consists of a single sweep over all nodes in the tree. Any ordering of the boxes in which a parent box is processed after its children can be used. For each leaf box τ, approximations $\mathbf{S}^{\tau}$ and $\mathbf{F}^{\tau}$ to the solution operators for the homogeneous and particular solutions are constructed. Additionally, approximations $\mathbf{T}^\tau$ and $\mathbf{H}^\tau$ to the local continuum operators that map boundary data and body load for a particular solution to the boundary fluxes of the resulting particular solution are constructed using the procedure described in Section 3.3. For a parent box τ with children α and β, we construct the solution operators $\mathbf{X}^\tau_{\text{gi,gi}}$ and $\mathbf{S}^\tau_{\text{gi,ge}}$, and the DtN operator $\mathbf{T}^\tau_{\text{ge,ge}}$ via the process described in Section 3.4. Algorithm 1 summarizes the build stage.

3.5.3 The solve stage

After the "build stage" described in Algorithm 1 has been completed, an approximation to the global solution operator of (3.1) has been computed, and represented through the various matrices ($\mathbf{H}^\tau$, $\mathbf{F}^\tau$, etc.) described in Section 3.5.2. Then given specific boundary data f and a body load g, the corresponding solution u to (3.1) can be found through a "solve stage" that involves two passes through the tree, first an upwards pass (from smaller to larger boxes), and then a downwards pass. In the upward pass, the particular solutions and normal derivatives of the particular solution are computed and stored in the vectors $\mathbf{w}$ and $\mathbf{h}$ respectively. Then by sweeping down the tree applying the solution operators $\mathbf{S}$ to the Dirichlet boundary data for each box τ and adding the particular solution, the approximate solution $\mathbf{u}$ is computed. Algorithm 2 summarizes the solve stage.

We observe that the vectors $\mathbf{w}^\tau_{\text{gi}}$ can all be stored on a global vector $\mathbf{w} \in \mathbb{R}^N$. Since each boundary collocation node $\mathbf{y}_j$ belongs to precisely one index vector I^τ_{gi}, we simply find that $\mathbf{w}^\tau_{\text{gi}} = \mathbf{w}(I^\tau_{\text{gi}})$.

Remark 3 (Efficient storage of particular solutions) For notational simplicity, we describe Algorithm 2 (the "solve stage") in a way that assumes that for each box τ, we explicitly store a corresponding vector $\mathbf{h}^\tau_{\text{ge}}$ that represents the boundary fluxes for the local particular solution. In practice, these vectors can all be stored on a global vector $\mathbf{h} \in \mathbb{R}^N$, in a manner similar to how we store $\mathbf{w}$. For any box τ with children α and β, we store on $\mathbf{h}$ the *difference* between the boundary fluxes, so that $\mathbf{h}(I^\tau_{\text{gi}}) = -\mathbf{h}^\alpha_3 + \mathbf{h}^\beta_3$. In other words, as soon as the boundary fluxes have been computed for a box α, we add its contributions to the vector $\mathbf{h}(I^\alpha_{\text{ge}})$ with the appropriate signs and then delete it. This becomes notationally less clear, but is actually simpler to code.

3.5.4 Algorithmic complexity

In this section, we determine the asymptotic complexity of the direct solver. The analysis is very similar to the analysis seen in [16] for no body load. Let $N_{\text{leaf}} = 4q$ denote the number of Gaussian nodes on the boundary of a leaf box, and let p^2 denote the number of Chebychev nodes used in the leaf computation. In the asymptotic analysis, we set $p = q + 2$, so that $p \sim q$. Let L denote the number of levels in the binary tree. This means there are 2^L boxes. Thus the total number of discretization nodes N is approximately $2^L q^2$.

In processing a leaf, the dominant cost involves matrix inversion (or factorization followed by triangular solve) and matrix-matrix multiplications. The largest matrices encountered are of size $O(q^2) \times O(q^2)$, making the cost to process one leaf $O(q^6)$. Since there are N/q^2 leaf boxes, the total cost of pre-computing approximate DtN operators for all the bottom level is $\sim (N/q^2) \times q^6 \sim N\,q^4$.

Next, consider the process of merging two boxes, as described in Section 3.4. On level ℓ, there are 2^ℓ boxes, that each have $O(2^{-\ell/2}N^{0.5}))$ nodes along their boundaries. (On level $\ell = 2$, there are 4 boxes that each have side length one half of the original side length; on level $\ell = 4$, there are 16 boxes that have side length one quarter of the original side length; etc.) The cost of executing a merge is dominated by the cost to perform matrix algebra (inversion, multiplication, etc) of dense matrices of size $2^{-\ell/2}N^{0.5} \times 2^{-\ell/2}N^{0.5}$. This makes the total cost for the merges in the upwards pass

$$\sum_{\ell=1}^{L} 2^\ell \times \left(2^{-\ell/2}N^{0.5}\right)^3 \sim \sum_{\ell=1}^{L} 2^\ell \times 2^{-3\ell/2}N^{1.5} \sim N^{1.5}\sum_{\ell=1}^{L} 2^{-\ell/2} \sim N^{1.5}.$$

Finally, consider the cost of the solve stage (Algorithm 2). We first apply at each of the 2^L leaves the operators $\mathsf{H}^{\tau}_{\text{ge,ci}}$, which are all of size $4q \times (p-2)^2$, making the overall cost $\sim 2^L q^3 = N\,q$ since $p \sim q$ and $N \sim 2^L q^2$. In the upwards sweep, we apply at level ℓ matrices of size

Algorithm 1 (Build stage for problems with body load)

This algorithm builds all solution operators required to solve the non-homogeneous BVP (3.1). It is assumed that if node τ is a parent of node σ, then $\tau < \sigma$.

```
for $\tau = N_{\text{boxes}},\, N_{\text{boxes}}-1,\, N_{\text{boxes}}-2,\, \dots,\, 1$
    if ($\tau$ is a leaf)
        $\mathsf{F}^{\tau}_{\text{c,ci}} = \begin{bmatrix} \mathbf{0} \\ \mathsf{A}^{-1}_{\text{ci,ci}} \end{bmatrix}$    [pot.] ← [body load]
        $\mathsf{H}^{\tau}_{\text{ge,ci}} = \mathsf{D}_{\text{ge,c}}\mathsf{F}^{\tau}_{\text{c,ci}}$    [deriv.] ← [body load]
        $\mathsf{S}^{\tau}_{\text{c,ge}} = \begin{bmatrix} \mathsf{I} \\ -\mathsf{A}^{-1}_{\text{ci,ci}}\mathsf{A}_{\text{ci,ce}} \end{bmatrix} \mathsf{L}_{\text{ce,ge}}$    [pot.] ← [pot.] (solution operator)
        $\mathsf{T}^{\tau}_{\text{ge,ge}} = \mathsf{D}_{\text{ge,c}}\mathsf{S}_{\text{c,ge}}$    [deriv.] ← [pot.] (DtN operator)
    else
        Let $\alpha$ and $\beta$ be the children of $\tau$.
        Partition $I^{\alpha}_{\text{ge}}$ and $I^{\beta}_{\text{ge}}$ into vectors $I_1$, $I_2$, and $I_3$ as shown in Figure 3.2.
        $\mathsf{X}^{\tau}_{\text{gi,gi}} = \left(\mathsf{T}^{\alpha}_{3,3} - \mathsf{T}^{\beta}_{3,3}\right)^{-1}$    [pot.] ← [deriv.]
        $\mathsf{S}^{\tau}_{\text{gi,ge}} = \mathsf{X}^{\tau}_{\text{gi,gi}}\left[-\mathsf{T}^{\alpha}_{3,1} \mid \mathsf{T}^{\beta}_{3,2}\right]$    [pot.] ← [pot.] (solution operator)
        $\mathsf{T}^{\tau}_{\text{ge,ge}} = \begin{bmatrix} \mathsf{T}^{\alpha}_{1,1} & \mathbf{0} \\ \mathbf{0} & \mathsf{T}^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} \mathsf{T}^{\alpha}_{1,3} \\ \mathsf{T}^{\beta}_{2,3} \end{bmatrix} \mathsf{S}^{\tau}_{\text{gi,ge}}$    [deriv.] ← [pot.] (DtN operator).
    end if
end for
```

Figure 3.3: Build stage.

ALGORITHM 2 (Solver for problems with body load)

This algorithm constructs an approximation $\mathbf{u}$ to the solution u of (3.1). It uses the matrices that represent the solution operator that were constructed using Algorithm 3. It is assumed that if node τ is a parent of node σ, then $\tau < \sigma$.

Upwards pass — construct all particular solutions:

```
for $\tau = N_{\mathrm{boxes}},\, N_{\mathrm{boxes}} - 1,\, N_{\mathrm{boxes}} - 2,\, \dots,\, 1$
    if ($\tau$ is a leaf)
        # Compute the boundary fluxes of the local particular solution.
        $\mathbf{h}^{\tau}_{\mathrm{ge}} = \mathbf{H}^{\tau}_{\mathrm{ge,ci}}\,\mathbf{g}^{\tau}_{\mathrm{ci}}$
    else
        Let $\alpha$ and $\beta$ denote the children of $\tau$.
        # Compute the local particular solution.
        $\mathbf{w}^{\tau}_{\mathrm{gi}} = \mathbf{X}^{\tau}_{\mathrm{gi,gi}}\left(-\mathbf{h}^{\alpha}_{3} + \mathbf{h}^{\beta}_{3}\right).$
        # Compute the boundary fluxes of the local particular solution.
        $\mathbf{h}^{\tau}_{\mathrm{ge}} = \begin{bmatrix} \mathbf{h}^{\alpha}_{1} \\ \mathbf{h}^{\beta}_{2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \mathbf{w}^{\tau}_{\mathrm{gi}}.$
    end if
end for
```

Downwards pass — construct all potentials:

```
# Use the provided Dirichlet data to set the solution on the exterior of the root.
$\mathbf{u}(I^{1}_{\mathrm{ge}}) = \{f(\boldsymbol{y}_k)\}_{k \in I^{1}_{\mathrm{ge}}}.$
for $\tau = 1,\, 2,\, 3,\, \dots,\, N_{\mathrm{boxes}}$
    if ($\tau$ is a parent)
        # Add the homogeneous term and the particular term.
        $\mathbf{u}^{\tau}_{\mathrm{gi}} = \mathbf{S}^{\tau}_{\mathrm{gi,ge}}\,\mathbf{u}^{\tau}_{\mathrm{ge}} + \mathbf{w}^{\tau}_{\mathrm{gi}}.$
    else
        # Add the homogeneous term and the particular term.
        $\mathbf{u}^{\tau}_{\mathrm{c}} = \mathbf{S}^{\tau}_{\mathrm{c,ge}}\,\mathbf{u}^{\tau}_{\mathrm{ge}} + \mathbf{F}^{\tau}_{\mathrm{c,ci}}\,\mathbf{g}^{\tau}_{\mathrm{ci}}.$
    end
end for
```

Figure 3.4: Solve stage.

$O(2^{-\ell/2}N^{0.5}) \times O(2^{-\ell/2}N^{0.5})$ on 2^ℓ boxes, adding up to an overall cost of

$$\sum_{\ell=1}^{L} 2^\ell \times \left(2^{-\ell/2}\,N^{0.5}\right)^2 \sim \sum_{\ell=1}^{L} 2^\ell \times 2^{-\ell}\,N \sim \sum_{\ell=1}^{L} N \sim N L \sim N \log N.$$

The cost of the downwards sweep is the same. However, the application of the matrices $\mathsf{F}^{\tau}_{\rm c,ci}$ at the leaves is more expensive since these are of size $O(q^2) \times O(q^2)$, which adds up to an overall cost of $2^L\,q^4 = N\,q^2$.

The analysis of the asymptotic storage requirements perfectly mirrors the analysis of the flop count for the solve stage, since each matrix that is stored is used precisely once in a matrix-vector multiplication. In consequence, the amount of storage required is

$$R \sim N\,q^2 + N\,\log N. \tag{3.23}$$

Remark 4 (A storage efficient version) The storage required for all solution operators can become prohibitive when the local order q is high, due to the term $N\,q^2$ in (3.23). One way to handle this problem is to not store the local solution operators for a leaf, but instead simply perform small dense solves each time the “solve stage” is executed. This makes the solve stage slower, obviously, but has the benefit of completely eliminating the Nq^2 term in (3.23). In fact, in this modified version, the overall storage required is $\sim NL \approx N\,\log_2(N/q^2)$, so we see that the storage costs decrease as q increases (as should be expected since we do all leaf computations from scratch in this case). Figure 3.9 provides numerical results illustrating the memory requirements of the various approaches.

3.6 Local refinement

When solving a boundary value problem like (3.1) it is common to have a localized loss of regularity due to, e.g., corners on the boundary, a locally non-smooth body load or boundary condition, or a localized loss of regularity in the coefficient functions in the differential operator. A common approach to efficiently regain high accuracy without excessively increasing the number of degrees of freedom used, is to locally refine the mesh near the troublesome location. In this manuscript, we assume the location is known and given, and that we manually specify the degree of local refinement. The difficulty that arises is that upon refinement, the collocation nodes on neighboring patches do not necessarily match up. To remedy this, interpolation operators are introduced to transfer information between patches. (The more difficult problem of determining how to automatically detect regions that require mesh refinement is a topic of current research.)

3.6.1 Refinement criterion

Suppose we desire to refine our discretization at some point $\hat{\boldsymbol{x}}$ in the computational domain (the point $\hat{\boldsymbol{x}}$ can be either in the interior or on the boundary). Consider as an example the situation depicted in Figure 3.5. For each level of refinement, we split any leaf box that contains $\hat{\boldsymbol{x}}$ and any "close" leaf boxes into a 2×2 grid of equal-sized leaf boxes. In Figure 3.5 we perform one level of refinement and find there are 6 leaf boxes "close" to $\hat{\boldsymbol{x}}$, which is represented by the green dot. These 6 boxes are refined into smaller leaf boxes.

A leaf box Ω_τ is close to $\hat{\boldsymbol{x}}$ if the distance d_τ from $\hat{\boldsymbol{x}}$ to the box Ω_τ satisfies $d_\tau \leq t l_\tau$, where $t = \sqrt{2}$ and $2l_\tau$ is the length of one side of the leaf box Ω_τ. In Figure 3.5 we show circles of size tl_γ and tl_β at the points in Ω_γ and Ω_β closest to $\hat{\boldsymbol{x}}$ (in this case the boxes are all the same size so $l_\gamma = l_\beta$). We see $\hat{\boldsymbol{x}}$ is "close" to Ω_γ, but not "close" to Ω_β. Just as in section 3.5.1, we place a $p \times p$ tensor product grid of Chebyshev nodes on each new leaf and a set of q Gaussian ("Legendre") interpolation nodes on the edge of each leaf. The vector $\{\boldsymbol{y}_j\}_{j=1}^N$ holds the locations of all Gaussian nodes across all leaves in the domain.

3.6.2 Refined mesh

Notice that with the refined grid the nodes along common boundaries are no longer aligned. Figure 3.6 is an example of such a grid. This is a problem during the build stage of the method since the merge operation is performed by equating the Neumann data on the common boundary. We begin the discussion on how to address this problem by establishing some notation.

Define two boxes as *neighbors* if they are on the same level of the tree and they are adjacent.

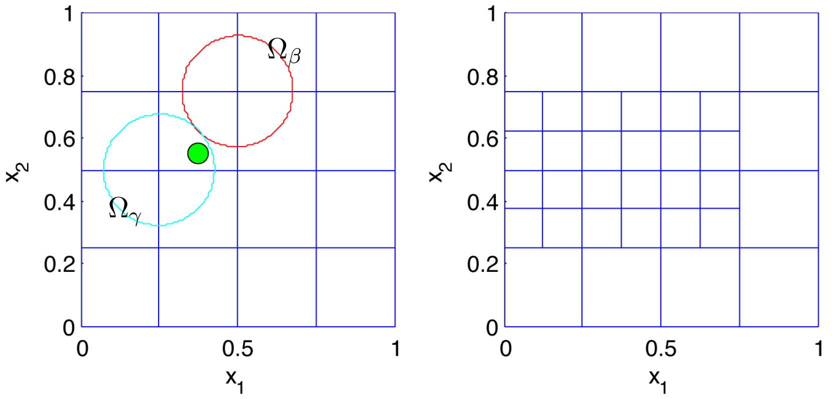
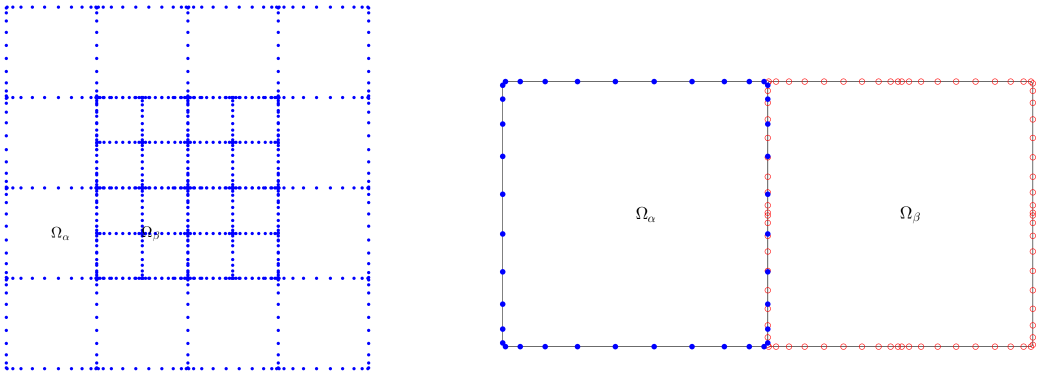


Figure 3.5: A sample domain where we desire to refine the grid at $\hat{\boldsymbol{x}}$, shown by the green circle. For leaf box Ω_γ the shortest distance to $\hat{\boldsymbol{x}}$ satisfies $d_\gamma < tl_\gamma$, so Ω_γ is refined. The maximum distance tl_γ is shown by the blue circle, which is centered at the closest point from Ω_γ to $\hat{\boldsymbol{x}}$. For leaf box Ω_β the shortest distance to $\hat{\boldsymbol{x}}$ does not satisfy $d_\beta < tl_\beta$, so Ω_β is not refined. The maximum distance tl_β is shown by the red circle, which is centered at the closest point from Ω_β to $\hat{\boldsymbol{x}}$.



(a)

(b)

Figure 3.6: (a) Grid with refinement at the center. (b) A close up of neighbors Ω_α and Ω_β. Since only one of the boxes is refined the exterior Gaussian nodes on the common boundary are not aligned.

In the case that only one of two neighbors has been refined, such as Ω_α and Ω_β in Figure 3.6, special attention needs to be paid to the nodes on the common boundary. In order to merge boxes with different number of Gaussian nodes on the common edge, interpolation operators will be required. The next section describes this process in detail.

Consider the nodes on the common boundary between the two leaf boxes Ω_α and Ω_β. Let q denote the number of Gaussian nodes on one side of each leaf. Let $\{J_{\alpha,i}\}_{i=1}^{q}$ denote the index vector for the common boundary nodes from Ω_α and $\{J_{\beta,i}\}_{i=1}^{2q}$ denote the index vector for the common boundary nodes from Ω_β. That is, recalling that $\mathbf{y}$ holds the locations of all Gaussian nodes in the domain, $\mathbf{y}(J_{\alpha,i})$ contains the q Gaussian nodes on the Eastern side of box Ω_α and $\mathbf{y}(\mathbf{J}_{\beta,i})$ contains the $2q$ nodes on the Western side of box Ω_β.

3.6.3 Modifications to build stage

Once the grid with the Gaussian and Chebyshev nodes is constructed, as described in section 3.6.2, the build stage starts with the construction of all leaf operators as described in Section 3.3. Then, for simplicity of presentation, boxes are merged from the lowest level moving up the tree. After merging the children of a refined parent, such as Ω_β in Figure 3.6 it is seen that the parent's exterior nodes do not align with the exterior nodes of any neighbor which has not been refined.

Recalling the index notation used in section 3.6.2, we form the interpolation matrix $\mathbf{P}_{\text{up,W}}$ mapping data on $\mathbf{y}(J_{\alpha,i})$ to data on $\mathbf{y}(J_{\beta,i})$ and the interpolation matrix $\mathbf{P}_{\text{down,W}}$ mapping data on $\mathbf{y}(\mathbf{J}_{\beta,i})$ to data on $\mathbf{y}(J_{\alpha,i})$. Observe that when interpolating from two sets of q Gaussian nodes to a set of q Gaussian nodes, the interpolation must be done as two separate interpolations from q to $q/2$ nodes. The matrix $\mathbf{P}_{\text{down,W}}$ is a block diagonal matrix consisting of two $q/2 \times q$ matrices (assuming q is divisible by 2).

For a refined parent, such as Ω_β in Figure 3.6, we form the operators $\mathbf{T}^\beta$, $\mathbf{S}^\beta$, and $\mathbf{X}^\beta$ and form the interpolation operators for every side of the parent, regardless of whether the exterior nodes align with the neighbor's exterior nodes. Observe that in the case of Ω_β in Figure 3.6 the Eastern and Northern sides of Ω_β will have $\mathbf{P}_{\text{down}} = \mathbf{P}_{\text{up}} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

Then interpolation operators mapping the entire boundary data between fine and coarse grids are given by block diagonal matrices $\mathbf{P}_{\text{up}}$ and $\mathbf{P}_{\text{down}}$ whose diagonal blocks are the interpolation operators for each edge. The interpolation operators for Ω_β are

$$\mathbf{P}^{\beta}_{\text{up}} = \texttt{blkdiag}(\mathbf{P}^{\beta}_{\text{up,S}}, \mathbf{P}^{\beta}_{\text{up,E}}, \mathbf{P}^{\beta}_{\text{up,N}}, \mathbf{P}^{\beta}_{\text{up,W}})$$

and

$$\mathbf{P}^{\beta}_{\text{down}} = \texttt{blkdiag}(\mathbf{P}^{\beta}_{\text{down,S}}, \mathbf{P}^{\beta}_{\text{down,E}}, \mathbf{P}^{\beta}_{\text{down,N}}, \mathbf{P}^{\beta}_{\text{down,W}}).$$

(The text `blkdiag` denotes the function that forms a block diagonal matrix from its arguments.) Then we form the new operators $\mathbf{T}^{\beta}_{\text{new}}$ and $\mathbf{S}^{\beta}_{\text{new}}$ for the parent box Ω_β as follows

$$\mathbf{T}^{\beta}_{\text{new}} = \mathbf{P}^{\beta}_{\text{down}} \mathbf{T}^{\beta} \mathbf{P}^{\beta}_{\text{up}}$$

and

$$\mathbf{S}^{\beta}_{\text{new}} = \mathbf{S}^{\beta} \mathbf{P}^{\beta}_{\text{up}}.$$

Now $\mathbf{T}^{\beta}_{\text{new}}$ is a map defined on the same set of points as all of the neighbors of Ω_β and Neumann data can be equated on all sides.

Next, suppose a refined parent does not have a neighbor on one of its sides. Then on that side we use $\mathbf{P}_{\text{down}} = \mathbf{P}_{\text{up}} = \mathbf{I}$. This could happen if the parent is on the boundary of our domain Ω. For example, suppose Ω_α in Figure 3.6 was also refined. Then Ω_α would not have a neighbor on its Western side. Additionally, if multiple levels of refinement are done then a refined parent could have no neighbors on one side. For example, suppose the Northwestern child of Ω_β in Figure 3.6 was refined. Then the Northwestern child would not have a neighbor on its Western side since box Ω_α is on a different level of the tree.

Forming the interpolation operators for each side of Ω_β before we perform any following merge operations is the easiest approach. The alternative would be to form an interpolation operator every time two boxes are merged and the nodes do not align.

3.6.4 Modifications to solve stage

On the upwards pass of the solve stage, the fluxes for the particular solution must be calculated on the same nodes so the particular solution can be calculated on those nodes. This is easily achieved by applying the already computed interpolation operator $\mathbf{P}_{\text{down}}$ to obtain $\mathbf{h}^{\beta}_{\text{new}} = \mathbf{P}_{\text{down}}\mathbf{h}^{\beta}_{\text{old}}$.

In the downwards pass of the solve stage, the application of the solution operators results in the approximate solution at the coarse nodes on the Western and Southern sides of Ω_β. The solution operator $\mathbf{S}^{\beta}_{\text{new}}$ now maps the solution on the coarse nodes on the Western and Southern sides of Ω_β (and the dense nodes on the Eastern and Northern sides) to the solution on the interior of Ω_β. However, we also need the solution on the dense nodes on the Western and Southern sides of Ω_β. Let $\mathbf{u}_{ge,\text{new}}$ denote the solution on the boundary of Ω_β with the coarse nodes on the Western and Southern edges. Then the approximate solution on the dense nodes is given by $\mathbf{u}_{ge,\text{old}} = \mathbf{P}_{\text{up}}\mathbf{u}_{ge,\text{new}}$.

3.6.5 Impact of refinement on the asymptotic complexity

We demonstrated in Section 3.5.4 that the asymptotic complexity of the build stage is $O(N^{3/2})$ for the case of a uniform balanced tree. It turns out that such a tree is the most adversarial case from the point of view of the number of flops required, just as for nested dissection and multifrontal solvers. While we will not carry out a detailed analysis of every possible tree, it is perhaps illustrative to consider what happens in the extreme case where we start with a course level grid consisting of 4×4 nodes, each with a $q \times q$ tensor product grid. Then we recursively refine the middle 2×2 nodes, as shown in Figure 3.7. In this case, each refinement adds roughly $4q^2$ nodes, so that with L levels in the tree, the total number of nodes satisfies $N \sim Lq^2$. During the build stage, the processing of each level in the tree requires dense operations to be carried out on matrices of size $O(q^2) \times O(q^2)$, at cost $O(q^6)$. This leads to a total cost of $T \sim L\,q^6 \sim N\,q^4$ for the case of local refinement around a single point. In the general case, asymptotic complexity between $O(N)$ and $O(N^{3/2})$, depending on the structure of the tree, will be observed.

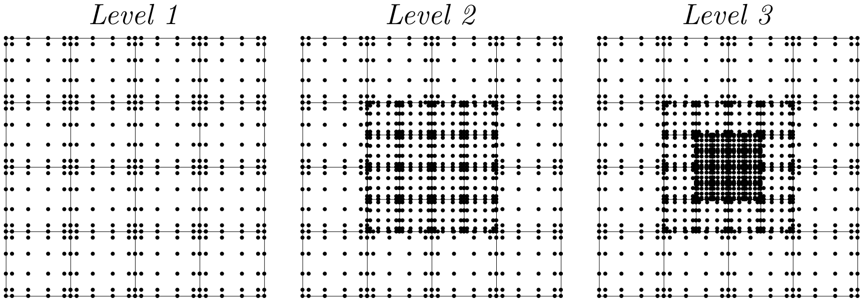


Figure 3.7: Local refinement around a single point, as discussed in Section 3.6.5. In the figure, each leaf has a $q \times q$ local grid, for $q = 6$. At each level of refinement, the center 2×2 block of nodes is refined into 4×4 blocks of half the size.

3.7 Numerical experiments

In this section, we present the results of numerical experiments that illustrate the performance of the scheme proposed. Section 3.7.1 reports on the computational cost and memory requirements. Sections 3.7.2–3.7.5 report on the accuracy of the proposed solution technique for a variety of problems where local mesh refinement is required. Finally, Section 3.7.6 illustrates the use of the proposed method in the acceleration of an implicit time stepping scheme for solving a parabolic partial differential equation.

For each experiment, the error is calculated by comparing the approximate solution with a reference solution $\mathbf{u}_{\text{ref}}$ constructed using a highly over resolved grid. Errors are measured in ℓ^{∞}-norm, on all Chebyshev nodes on leaf boundaries.

In all of the experiments, each leaf is discretized using a $p \times p$ tensor product mesh of Chebyshev nodes. The number of Legendre nodes per leaf edge is set to $q = p-1$. In all experiments except the one described in Section 3.7.5, the computational domain is the square $\Omega = [0,1]^2$ discretized into $n \times n$ leaf boxes, making the total number of degrees of freedom roughly $N \approx p^2 \times n^2$ (to be precise, $N = n^2 \times (p-1)^2 + 2n \times (p-1) + 1$).

The proposed method was implemented in Matlab and all experiments were run on a laptop computer with a 4 core Intel i7-3632QM CPU running at 2.20 GHz with 12 GB of RAM.

3.7.1 Computational speed

The experiments in this section illustrate the computational complexity and memory requirements of the direct solver. Recall that the asymptotic complexity of the method scales as nested dissection or multifrontal methods, with execution times scaling as $O(N^{3/2})$ and $O(N \log N)$ for the "build" and "solve" stages, respectively. The asymptotic memory requirement is $O(N \log N)$.

The computational complexity and memory requirements of the proposed method depend only on the domain and the computational mesh; the choice of PDE is irrelevant. In the experiments reported here, we used $\Omega = [0,1]^2$ with a uniform mesh.

Figure 3.8 reports the time in seconds for the (a) build and (b) solve stages of the proposed solution technique when there is a body load (BL), when the leaf computation is done on the fly as described in Remark 4 (BL(econ)), and when there is no body load (NBL). Results for two different orders of discretization ($q = 8$ and 16) are shown. Notice that as expected the constant scaling factor for both stages is larger for the higher order discretization. (It may be of interest that our numerical experiments in many cases demonstrate *linear* scaling for the build stage, despite its $O(N^{3/2})$ asymptotic algorithmic complexity. The reason is that even for $N \sim 10^6$, we have hardly yet entered the regime where the $O(N^{3/2})$ term starts to dominate.)

Figure 3.9 reports on memory requirements. Letting R denote total memory used, we plot R/N versus the number of discretization points N, where R is measured in terms of number of floating point numbers. We see that storing the solution operators on the leaves is quite costly in terms of memory requirements. The trade-off to be considered here is whether the main priority is to conserve memory, or to maximize the speed of the solve stage, cf. Remark 4. As an illustration, we see that for a problem with $q = 8$ and $n = 128$, for a total of 10^6 unknowns, the solve stage takes 1.7 seconds with the solution operators stored versus 9.8 seconds for performing the local solves on the fly. In situations where the solution is only desired in prescribed local regions of the geometry, computing the operators on the fly is ideal.

Remark 5 (Complexity for a locally refined tree) We claim in Section 3.6.5 that for a locally refined tree, the complexity of the scheme is no worse (and sometimes better) than the $O(N^{3/2})$ complexity obtained for a balanced uniform tree. To substantiate this claim, we timed an experiment involving a locally refined tree, as shown in Figure 3.7. The results are presented in Figure 3.10, for two different orders of discretization ($q = 16$ and 32). We use larger numbers for q in this section so that we obtain a significant increase in the number of nodes. If we use $q = 8$ then each refinement only adds a few hundred nodes and it is hard to see any significant scaling without doing an unreasonable number of refinements. For reference, we include a comparison to the case of $q = 16$ with a body load and no refinement.

Remark 6 When the underlying BVP that is discretized involves a constant coefficient operator, many of the leaf solution operators are identical. This observation can be used to greatly reduce storage requirements while maintaining very high speed in the solve stage. This potential acceleration was *not* exploited in the numerical experiments reported.

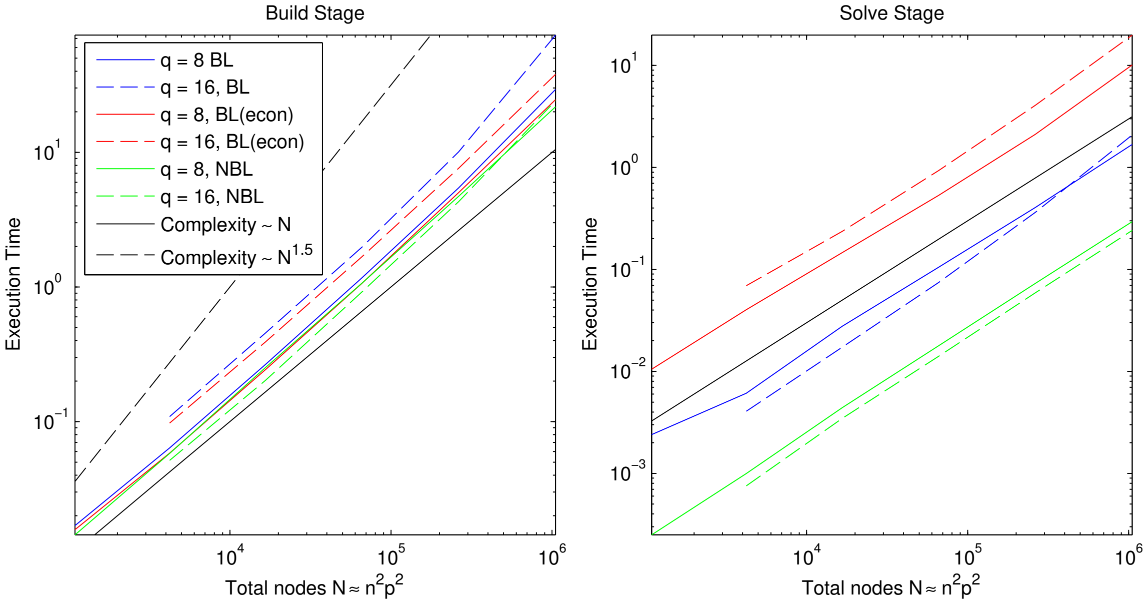
3.7.2 Variable coefficients

In this section, the proposed scheme is applied to the variable coefficient Helmholtz problem

$$-\Delta u - \kappa^2(1 - c(\boldsymbol{x}))u = g, \qquad \boldsymbol{x} \in \Omega,$$

where $\Omega = [0,1] \times [0,1]$ and where c is a "scattering potential." The body load is taken to be a Gaussian given by $g = \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2)$ with $\alpha = 300$ and $\hat{\boldsymbol{x}} = [1/4, 3/4]$ while the variable coefficient is a sum of Gaussians $c(\boldsymbol{x}) = \frac{1}{2}\exp(-\alpha_2|\boldsymbol{x} - \hat{\boldsymbol{x}}_2|^2) + \frac{1}{2}\exp(-\alpha_3|\boldsymbol{x} - \hat{\boldsymbol{x}}_3|^2)$ with $\alpha_2 = \alpha_3 = 200$, $\hat{\boldsymbol{x}}_2 = [7/20, 6/10]$, and $\hat{\boldsymbol{x}}_3 = [6/10, 9/20]$ for the scattering potential. We set $\kappa = 40$, making the domain 6.4×6.4 wavelengths in size.

Figure 3.11 reports the l^∞ error versus the number of discretization points N. We get no accuracy for $q = 4$, but as q is increased, the errors rapidly decrease.



(a) (b)

Figure 3.8: (a) Time to execute build stage for the algorithm with and without a body load. These algorithms all have complexity $O(N^{1.5})$, and we see that the scaling factors depend strongly on the order of the method, but only weakly on whether body loads are included or not. (b) Time to execute the solve stage. Three cases are considered: NBL is the scheme for problems without a body load. BL is the scheme for problems with a body load. BL(econ) is a scheme that allows for body loads, but do not store the relevant solution operators at the leaves. p denotes the order in the local Chebyshev grids, and $q = p - 1$ is the number of Legendre nodes on the edge of each leaf.

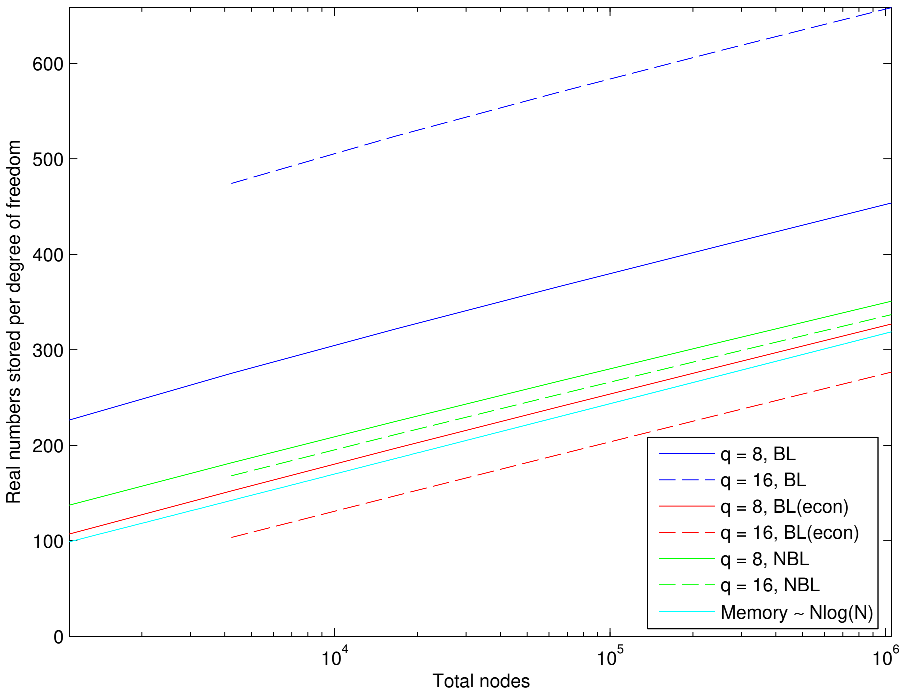
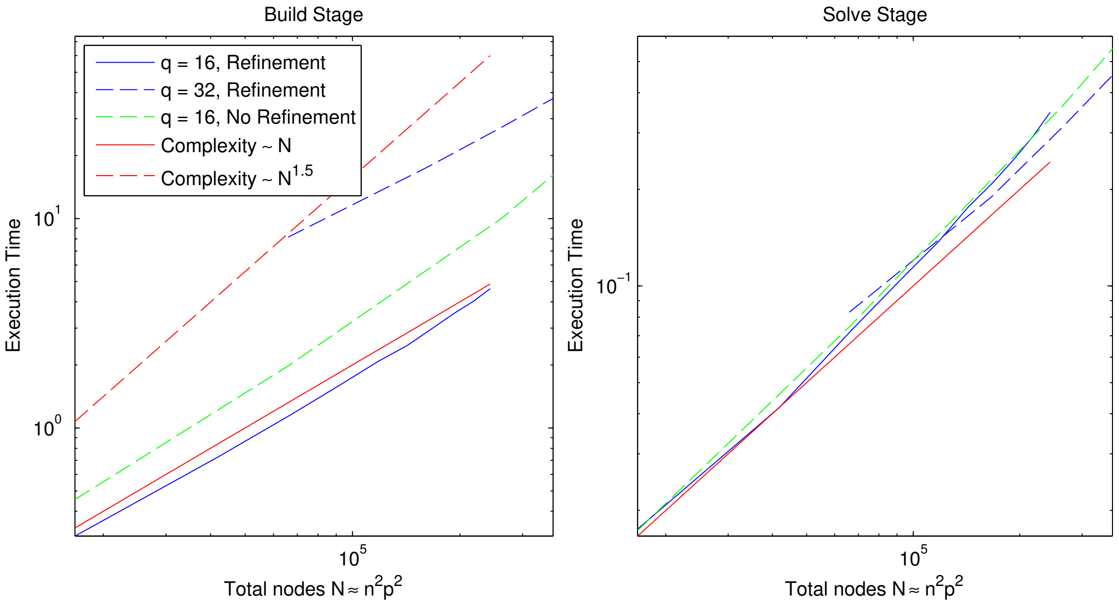


Figure 3.9: Memory requirements. Notation is as in Figure 3.8.



(a)

(b)

Figure 3.10: (a) Time to execute build stage for the algorithm with refinement. (b) Time to execute the solve stage with refinement. In both plots we see that the refinement does not impact the time to build the operators.

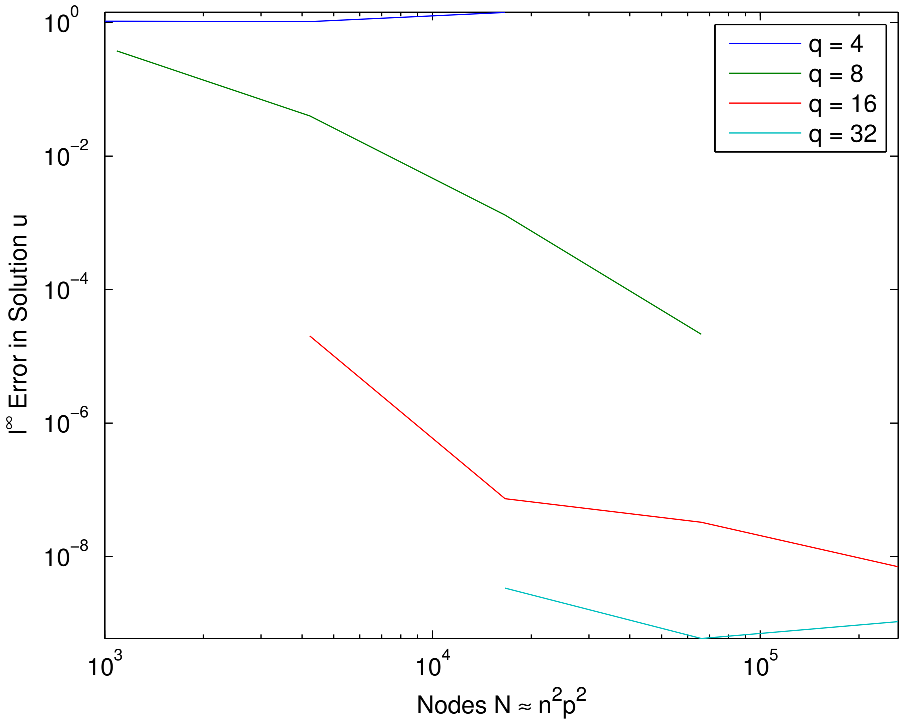


Figure 3.11: The error for the variable coefficient problem described in Section 3.7.2. As before, q denotes the number of Legendre nodes along one side of a leaf.

3.7.3 Concentrated body load

In this section, we consider a low frequency ($\kappa = 20$) Helmholtz boundary value problem

$$-\Delta u - \kappa^2 u = g, \qquad \boldsymbol{x} \in \Omega,$$

with $\Omega = [0,1] \times [0,1]$ and a very concentrated Gaussian for the body load, $g = \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2)$ with $\alpha = 3000$. In this case, we chose the Dirichlet boundary data to equal the solution to the free space equation $-\Delta u - \kappa^2 u = g$ with a radiation condition at infinity. In other words, u is the convolution between g and the free space fundamental solution. We computed the boundary data and the reference solution by numerically evaluating this convolution to very high accuracy.

To test the refinement strategy, we build a tree first with a uniform grid, i.e. $n \times n$ leaf boxes then add n_{ref} levels of refinement around the point $\hat{\mathbf{x}}$. Figure 3.12 reports the l^∞ norm of the error versus n_{ref} for four choices of uniform starting discretization. When $n = 4$ one level of refinement (i.e. 28 leaf boxes) results in approximately the same accuracy as when $n = 8$ and no levels of refinement (i.e. 64 leaf boxes).

3.7.4 Discontinuous body load

In section, we consider a Poisson boundary value problem on $\Omega = [0,1]^2$ with an indicator function body load g that has support $[1/4, 1/2] \times [1/4, 1/2]$. Observe that the lines of discontinuity of g coincide with edges of leaves in the discretization. Figure 3.13 reports the l^∞ error versus the number of discretization points N with uniform refinement for four different orders of discretization. Note that the approximate solution and its first derivative are continuous through the boundaries of the leaves (even on the boundaries where the jump in the body load occurs) since the algorithm enforces them by derivation.

Remark 7 Applying the scheme to a problem where a discontinuity in the body load does *not* align with the leaf boundaries results in a very low accuracy approximation to the solution. For a problem analogous to the one described in this section, we observed slow convergence and attained no better than two or three digits of accuracy on the most finely resolved mesh.

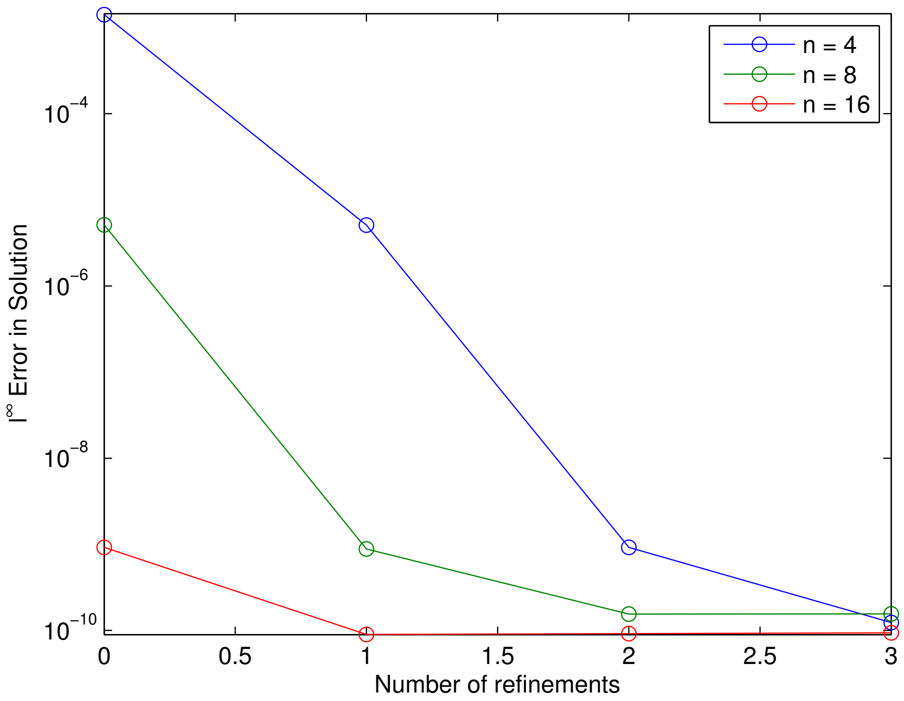


Figure 3.12: Error for Helmholtz equation with $\kappa = 20$ and a very concentrated body load, demonstrating the ability to improve the solution with refinement. We use local Chebyshev grids with 17×17 Chebyshev nodes per leaf, and $n \times n$ leaves, before refinement. For a problem like this with a concentrated body load we can improve the error just as much by refining the discretization at the troublesome location as we can from doubling the number of leaves, which would give the same grid at the target location.

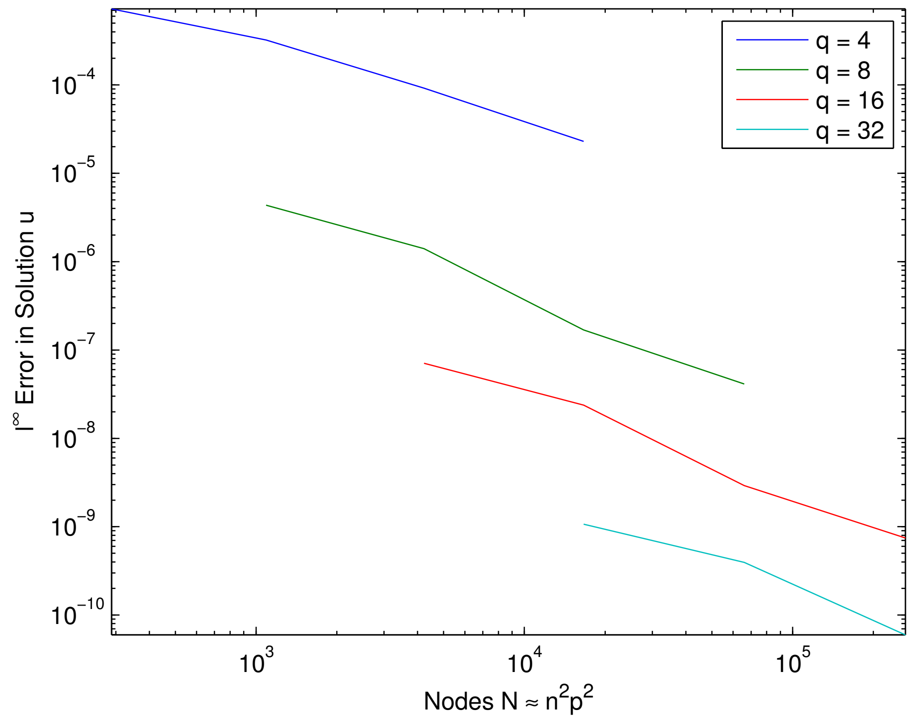


Figure 3.13: The error for a problem with a discontinuous body load. The discontinuities align with the edges of the leaves so we still get 10 digits of accuracy. In the legend, q denotes the number of Legendre nodes along one side of a leaf.

3.7.5 Tunnel

This section reports on the performance of the solution technique when applied to the Helmholtz Dirichlet boundary value problem

$$\begin{aligned} -\Delta u - \kappa^2 u = g, &\qquad \boldsymbol{x} \in \Omega, \\ u = f &\qquad \boldsymbol{x} \in \partial\Omega \end{aligned}$$

with $\kappa = 60$ where the domain Ω is a "tunnel" as illustrated in Figure 3.14. The body load is taken to be a Gaussian $g = \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2)$ with $\alpha = 300$ and $\hat{\boldsymbol{x}} = [1, 3/4]$. The Dirichlet boundary data is given by

$$f(\boldsymbol{x}) = \begin{cases} 0 & \text{for } x_1 \neq \pm 3 \\ \frac{1}{100}\sin(2\pi(x_2 - 3)) & \text{for } x_1 = 3 \\ \frac{1}{100}\sin(\pi(x_2 - 3)) & \text{for } x_1 = -3. \end{cases}$$

Note that $f(\boldsymbol{x})$ is continuous on $\partial\Omega$ and with this choice of wave number κ the domain Ω is about 10 wavelengths wide and 115 wavelengths long. The presence of the re-entrant corners results in a solution that has strong singularities which require local refinement in order for the method to achieve high accuracy.

Figure 3.15 reports the l^∞ error versus the number of refinements into the corners with three choices of coarse grid. We use $q = 16$ for all examples and h gives the width and height of each leaf box. When $h = 1/4$, the discretization is only sufficient to resolve the Helmholtz equation with $\kappa = 60$ within 1% of the exact solution. When $h = 1/8$, the solution technique stalls at 5 digits of accuracy independent of the number of refinement levels.

Remark 8 (Symmetries) This problem is rich in symmetries that can be used to accelerate the build stage. In our implementation, we chose to exploit the fact that the tunnel is made up of four L-shaped pieces glued together. The DtN operator and corresponding solution operators were constructed for one L-shape. Then creating the solver for the entire geometry involved simply gluing the 4 L-shaped geometries together via *three* merge operations.

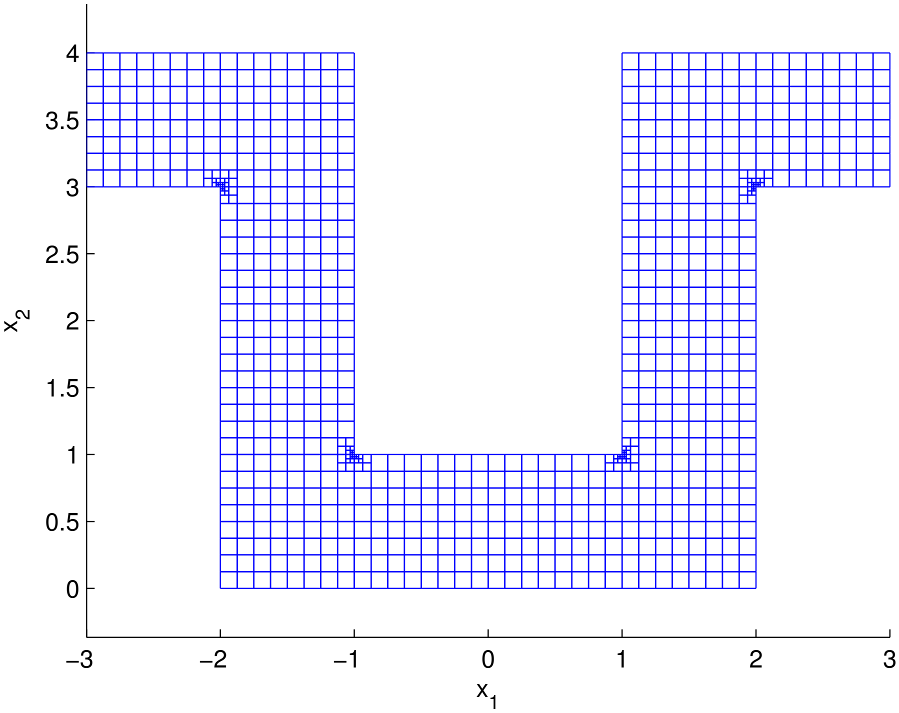


Figure 3.14: The domain used for the tunnel problem. We solve Helmholtz equation with $\kappa = 60$, making the tunnel about 10λ wide and 115λ long. The end caps have fixed Dirichlet data and the sides of the tunnel have the Dirichlet data set to $u(\boldsymbol{x}) = 0$.

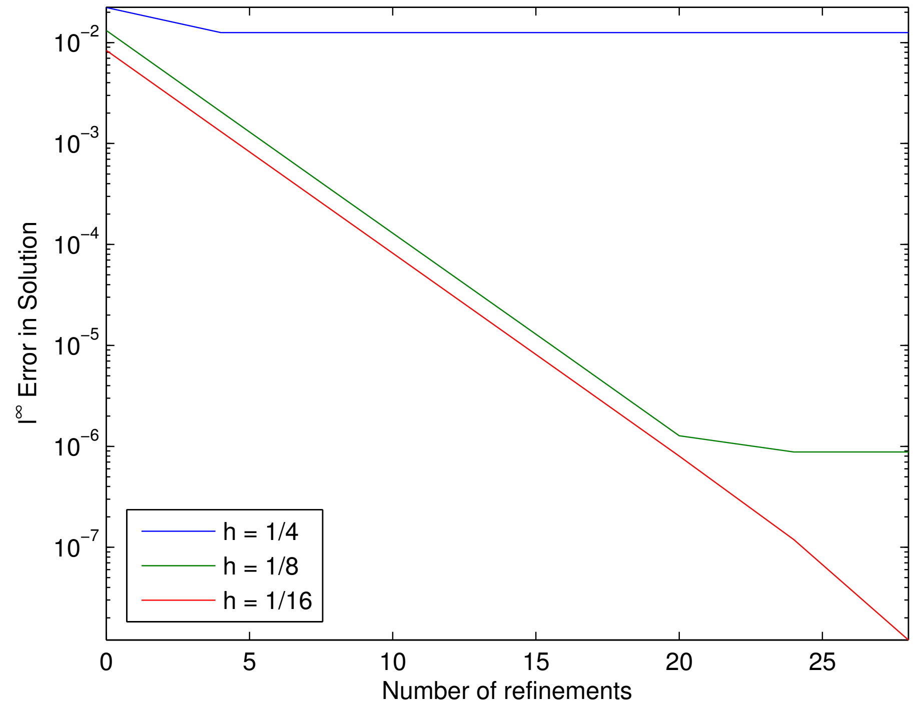


Figure 3.15: Error for Helmholtz equation with $\kappa = 60$ on the tunnel. The end caps have fixed Dirichlet data and the sides of the tunnel have the Dirichlet data set to $f(\boldsymbol{x}) = 0$. A Gaussian $g = \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2)$ with $\alpha = 300$ located at $\hat{\boldsymbol{x}} = [1, 3/4]$ is used for the body load.

3.7.6 A parabolic problem

Our final numerical example involves a convection-diffusion initial value problem on $\Omega = [0,1]^2$ given by

$$\begin{aligned} \Big(\epsilon\Delta - \frac{\partial}{\partial x_1}\Big)u(\boldsymbol{x},t) &= \frac{\partial u}{\partial t}, && \boldsymbol{x} \in \Omega,\ t > 0 \\ u(\boldsymbol{x},0) &= \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2), && \boldsymbol{x} \in \Omega. \end{aligned}$$

We imposed zero Neumann boundary conditions on the south and north boundaries ($x_2 = 0,1$) and periodic boundary conditions on the west and east boundaries ($x_1 = 0,1$). These boundary conditions correspond to fluid flowing through a periodic channel where no fluid can exit the top or bottom of the channel. To have a convection dominated problem, we chose $\epsilon = 1/200$. Finally, the parameters in the body load were chosen to be $\alpha = 50$ and $\hat{\boldsymbol{x}} = [1/4, 1/4]$.

Applying the Crank-Nicolson time stepping scheme with a time step size k results in having to solve the following elliptic problem at each time step:

$$\Big(\frac{1}{k}I - \frac{1}{2}A\Big)u_{n+1} = \Big(\frac{1}{k}I + \frac{1}{2}A\Big)u_n, \tag{3.24}$$

where $A = \epsilon\Delta - \partial/\partial x_1$ is our partial differential operator.

Observe that the algorithm does not change for this problem. The build stage execution time and memory requirement are identical to those seen in Section 3.7.1. The execution time for the solve stage for each individual time step is nearly identical to the solve stage execution time shown in Section 3.7.1. The only new step in the solve stage is the need to evaluate $(I/k + A/2)u_n$ at each time step.

Figure 3.16 reports the l^∞ error vs. the time step size k at three different times $t = 0.025,\ 0.1$, and 0.5. We use 16 leaf boxes per side with $q = 16$, to ensure that the spatial resolution error is far smaller than the time-stepping error. Note that even with a low order time stepping scheme, a high accuracy can still be attained since our fast time stepper allows us to use a very short time step without incurring an unduly long solution time.

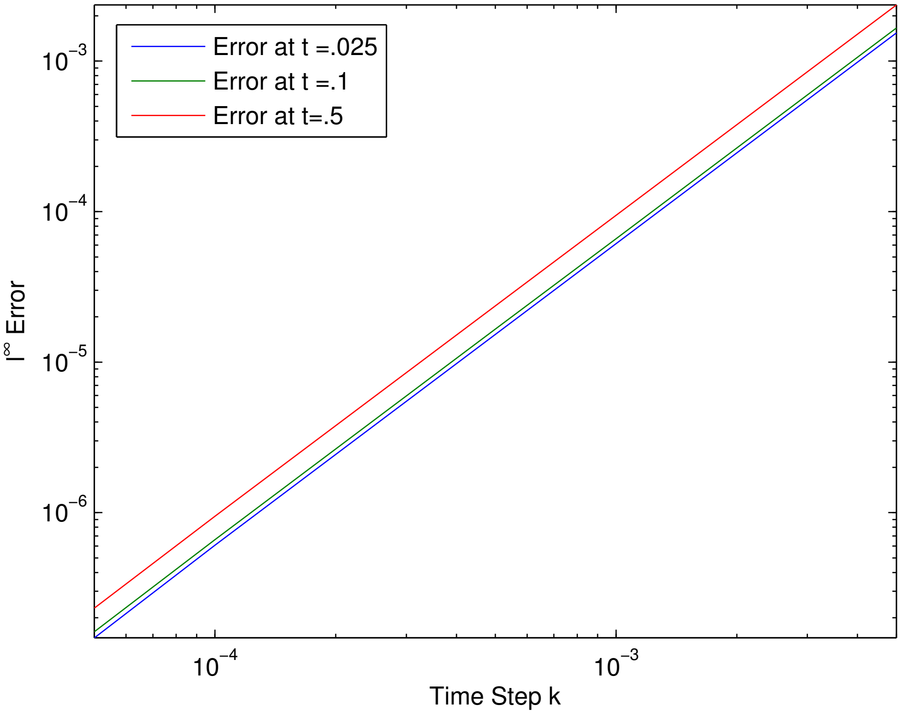


Figure 3.16: Error for the convection-diffusion equation described in Section 3.7.6. The error is estimated by comparing against a highly over-resolved solution.

3.8 Concluding remarks

We have described an algorithm for solving non-homogeneous linear elliptic PDEs in two dimensions based on a multidomain spectral collocation discretization. The solver is designed explicitly for being combined with a nested dissection type direct solver. Its primary advantage over existing methods is that it enables the use of very high order local discretization without jeopardizing computational efficiency in the direct solver. The scheme is an evolution on previously published methods [29, 15, 16]. The novelty in this work is that the scheme has been extended to allow for problems involving body loads, and for local refinement.

The scheme is particularly well suited for executing the elliptic solve required solving parabolic problems using implicit time-stepping techniques in situations where the domain is fixed, so that the elliptic solve is the same in every time step. In this environment, the cost of computing an explicit solution operator is amortized over many time-steps, and can also be recycled when the same equation is solved for different initial conditions.

The fact that the method can with ease incorporate high order local discretizations, and allows for very efficient implicit time-stepping appears to make it particularly well suited for solving the Navier-Stokes equations at low Reynolds numbers. Such a solver is currently under development and will be reported in future publications. Other extensions currently under way includes the development of adaptive refinement criteria (as opposed to the supervised adaptivity used in this work), and the extension to problems in three dimensions, analogous to the work in [23] for homogeneous equations.

Acknowledgements: The research reported was supported by DARPA, under the contract N66001-13-1-4050, and by the NSF, under the contracts DMS-1407340 and DMS-1620472.

Chapter 4

HPS Accelerated Spectral Solvers for Time Dependent Problems

Note: The material in this chapter is based on two manuscripts (with authors T. Babb, D. Appelö, P. G. Martinsson) that are currently in the review for the proceedings volume for the 2018 ICOSAHOM conference in London. The chapter presented here is an extended version with additional technical details and numerical examples.

Abstract: A high-order convergent numerical method for solving linear and non-linear parabolic PDEs is presented. The time-stepping is done via an explicit, singly diagonally implicit Runge-Kutta (ESDIRK) method of order 4 or 5, and for the implicit solve, we use the recently developed “Hierarchial Poincaré-Steklov (HPS)” method. The HPS method combines a multidomain spectral collocation discretization technique (a “patching method”) with a nested-dissection type direct solver. In the context under consideration, the elliptic solve required in each time-step involves the same coefficient matrix, which makes the use of a direct solver particularly effective. The manuscript describes the methodology and presents numerical experiments.

4.1 Introduction

This manuscript describes a highly computationally efficient solver for equations of the form

$$\kappa \frac{\partial u}{\partial t} = \mathcal{L}u(\boldsymbol{x}, t) + h(u, \boldsymbol{x}, t), \quad \boldsymbol{x} \in \Omega, t > 0, \tag{4.1}$$

with initial data $u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x})$. Here $\mathcal{L}$ is an elliptic operator acting on a fixed domain Ω and h is lower order, possibly nonlinear terms. We take κ to be real or imaginary, allowing for parabolic and Schrödinger type equations. We desire the benefits that can be gained from an implicit solver, such as L-stability and stiff accuracy, which means that the computational bottleneck will be the solution of a sequence of elliptic equations set on Ω. In situations where the elliptic equation to be solved is the same in each time-step, it is highly advantageous to use a *direct* (as opposed to *iterative*) solver. In a direct solver, an approximate solution operator to the elliptic equation is built once. The cost to build it is typically higher than the cost required for a single elliptic solve using an iterative method such as multigrid, but the upside is that after it has been built, each subsequent solve is very fast. In this manuscript, we argue that a particularly efficient direct solver to use in this context is a method obtained by combining a multidomain spectral collocation discretization (a so-called "patching method", see e.g. Ch. 5.13 in [5]) with a nested dissection type solver. It has recently been demonstrated [2, 16, 29] that this combined scheme, which we refer to as a "Hierarchial Poincaré-Steklov (HPS)" solver, can be used with very high local discretization orders (up to $p = 20$ or higher) without jeopardizing either speed or stability, as compared to lower order methods.

In this manuscript, we investigate the stability and accuracy that is obtained when combining high-order time-stepping schemes with the HPS method for solving elliptic equations. We restrict attention to relatively simple geometries (mostly rectangles). The method can without substantial difficulty be generalized to domains that can naturally be expressed as a union of rectangles, possibly mapped via curvilinear smooth parameter maps.

4.2 The Hierarchical Poincaré-Steklov Method

In this section, we describe a computationally efficient and highly accurate technique for solving an elliptic PDE of the form

$$\begin{cases} [Au](\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma, \end{cases} \tag{4.2}$$

where Ω is a domain with boundary Γ, and where A is a variable coefficient elliptic differential operator

$$[Au](\boldsymbol{x}) = -c_{11}(\boldsymbol{x})[\partial_1^2 u](\boldsymbol{x}) - 2c_{12}(\boldsymbol{x})[\partial_1\partial_2 u](\boldsymbol{x}) - c_{22}(\boldsymbol{x})[\partial_2^2 u](\boldsymbol{x}) \\ + c_1(\boldsymbol{x})[\partial_1 u](\boldsymbol{x}) + c_2(\boldsymbol{x})[\partial_2 u](\boldsymbol{x}) + c(\boldsymbol{x})\, u(\boldsymbol{x})$$

with smooth coefficients. In the present context, (4.2) represents an elliptic solve that is required in an implicit time-descretization technique of a parabolic PDE, as discussed in Section 4.1. For simplicity, let us temporarily suppose that the domain Ω is rectangular; the extension to more general domains is discussed in Remark 9.

Our ambition here is merely to provide a high level description of the method; for implementation details, we refer to [2, 15, 16, 23, 29, 30].

4.2.1 Discretization

We split the domain Ω into $n_1 \times n_2$ boxes, each of size $h \times h$. Then on each box, we place a $p \times p$ tensor product grid of Chebyshev nodes, as shown in Figure 4.1. We use collocation to discretize the PDE (4.2). With $\{\boldsymbol{x}_i\}_{i=1}^N$ denoting the collocation points, the vector $\mathbf{u}$ that represents our approximation to the solution u of (4.2) is given simply by $\mathbf{u}(i) \approx u(\boldsymbol{x}_i)$. We then discretize (4.2) as follows:

(1) For each collocation node that is *internal* to a box (red nodes in Figure 4.1), we enforce (4.2) by directly collocating the spectral differential operator supported on the box, as described in, e.g., Trefethen [37].

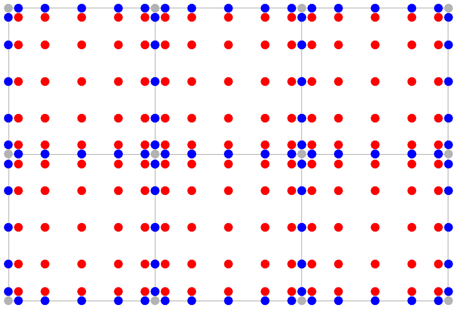


Figure 4.1: The domain Ω is split into $n_1 \times n_2$ squares, each of size $h \times h$. In the figure, $n_1 = 3$ and $n_2 = 2$. Then on each box, a $p \times p$ tensor product grid of Chebyshev nodes is placed, shown for $p = 7$. At red nodes, the PDE (4.2) is enforced via collocation of the spectral differentiation matrix. At the blue nodes, we enforce continuity of the normal fluxes. Observe that the corner nodes (gray) are excluded from consideration.

(2) For each collocation node on an *edge* between two boxes (blue nodes in Figure 4.1), we enforce that the normal fluxes across the edge be continuous. For instance, for a node $\boldsymbol{x}_i$ on a vertical line, we enforce that $\partial u/\partial x_1$ is continuous across the edge by equating the values for $\partial u/\partial x_1$ obtained by spectral differentiation of the boxes to the left and to the right of the edge. For an edge node that lies on the external boundary Γ, simply evaluate the normal derivative at the node, as obtained by spectral differentiation in the box that holds the node.

(3) All *corner* nodes (gray in Figure 4.1) are dropped from consideration. By a happy coincidence, it turns out that these values do not contribute to any of the spectral derivatives on the remaining nodes [13].

Since we exclude the corner nodes from consideration, the total number of nodes in the grid equals $N = (p-2)\big(p\,n_1\,n_2 + n_1 + n_2\big) \approx p^2\,n_1\,n_2$. The discretization procedure described then results in an $N \times N$ matrix A. For a node i, the value of $\mathsf{A}(i,:)\mathbf{u}$ depends on what type of node i

is:

$$\mathbf{A}(i,:)\mathbf{u} \approx \begin{cases} [Au](\boldsymbol{x}_i) \text{ for any interior (red) node,} \\ 0 \text{ for any edge node (blue) not on } \Gamma, \\ \partial u/\partial n \text{ for any edge node (blue) on } \Gamma. \end{cases}$$

This matrix $\mathbf{A}$ can be used to solve BVPs with a variety of different boundary conditions, including Dirichlet, Neumann, Robin, and periodic [29].

In many situations, a simple uniform mesh of the type shown in Figure 4.1 is not optimal, since the regularity in the solution may vary greatly, due to corner singularities, localized loads, etc. The HPS method can easily be adapted to handle local refinement. The essential difficulty that arises is that when boxes of different sizes are joined, the collocation nodes along the joint boundary will not align. It is demonstrated in [2, 13] that this difficulty can stably and efficiently be handled by incorporating local interpolation operators.

4.2.2 Corner Nodes

To solve the heat equation we use the spectral derivatives u_{xx} and u_{yy} in the interior of the leaf. However, at any grid point (x_0, y_0) these spectral derivatives will only depend on the values of u at the grid points along the line $y = y_0$ and $x = x_0$. Therefore, the values of u_{xx} and u_{yy} in the interior do not depend on the values of u at the four corner points. Furthermore, by the same reasoning, the normal derivative at a non-corner boundary point will not depend on the value of u at the four corner points.

Thus, the value of u at the corners has no effect on these other values. Therefore, the equations enforced in the HPS solver never depend on the values of u at the corner nodes and we can find the unique solution to the HPS scheme without ever considering the value of u at the corners and we therefore do not include them in the discretization when considering rectilinear grids.

It is worth noting that the values of cross derivatives u_{xy} will depend on the corner nodes. Such derivatives may be present in the PDE itself or can arise when using a curvilinear mapping from a non-rectangular domain to a rectangular domain. In such cases we include u at the corners in the discretization.

4.2.3 A Hierarchical Direct Solver

A key observation in previous work on the HPS method is that the sparse linear system that results from the discretization technique described in Section 4.2.1 is particularly well suited for direct solvers, such as the well-known multifrontal solvers that compute an LU-factorization of a sparse matrix. The key is to minimize fill-in by using a so called nested dissection ordering [14, 9]. Such direct solvers are very powerful in a situation where a sequence of linear systems with the same coefficient matrix needs to be solved, since each solve is very fast once the coefficient matrix has been factorized. This is precisely the environment under consideration here. The particular advantage of combining the multidomain spectral collocation discretization described in Section 4.2.1 is that the time required for factorizing the matrix is *independent* of the local discretization order. As we will see in the numerical experiments, this enables us to attain both very high accuracy, and very high computational efficiency.

Remark 9 (General domains) For simplicity we restrict attention to rectangular domains in this manuscript. The extension to domains that can be mapped to a union of rectangles via smooth coordinate maps is relatively straight-forward, since the method can handle variable coefficient operators [29, Sec. 6.4]. Some care must be exercised since singularities may arise at intersections of parameter maps, which may require local refinement to maintain high accuracy.

The direct solver described exactly mimics the classical nested dissection method, and has the same asymptotic complexity of $O(N^{1.5})$ for the "build" (or "factorization") stage, and then $O(N \log N)$ cost for solving a system once the coefficient matrix has been factorized. Storage requirements are also $O(N \log N)$. A more precise analysis of the complexity that takes into account the dependence on the order p of the local discretization shows [2] that $T_{\text{build}} \sim N\,p^4 + N^{1.5}$, and $T_{\text{solve}} \sim N\,p^2 + N \log N$.

4.2.4 HPS Algorithm

Previous implementations of the HPS algorithm have always enforced the PDE on the interior nodes and continuity of the derivative across leaf boundaries. However, previous implementations have utilized slightly different nodes on the leaf boundaries. In one case we utilized all Chebyshev nodes, including the corner nodes. However, as detailed in the previous section, the solution on all non-corner nodes has no dependence on the solution at the corner nodes and we can drop these. When the corner nodes are utilized we need to track the DtN operators acting in the x_1 and x_2 directions at the corner nodes. Dropping the corner nodes offers a simpler algorithm.

Another implementation had Gaussian nodes on the boundaries. These were originally used to avoid corner nodes where we had to track DtN operators in two directions at the corner nodes. However, we still utilized Chebyshev nodes on each individual leaf. This meant we had to find the solution and DtN operators on each leaf and then form interpolation operators mapping back and forth from the exterior Chebyshev nodes to the exterior Gaussian nodes.

The algorithm presented here, utilizing Chebyshev nodes and dropping the corners, enforces the same equations and is almost identical to the algorithms presented for the other discretizations. The only differences are that the presentation of the DtN operator on the exterior of each leaf is simpler and we drop the interpolation operators from Chebyshev to Gaussian nodes. The algorithm for this discretization is given below in Figures 4.2 and 4.3. The derivations and explanation of the operators along with the algorithms can be found for the case including Gaussian nodes in [2].

4.3 Time-stepping Methods

For high-order time-stepping of (4.1), we use the so called Explicit, Singly Diagonally Implicit Runge-Kutta (ESDIRK) methods. These methods have a Butcher diagram with a constant diagonal γ and are of the form

ALGORITHM 1 (Build stage for problems with body load)

This algorithms build all solution operators required to solve the non-homogeneous BVP (4.2).
It is assumed that if node τ is a parent of node σ, then $\tau < \sigma$.

```
for $\tau = N_{\text{boxes}},\, N_{\text{boxes}} - 1,\, N_{\text{boxes}} - 2,\, \dots,\, 1$
    if ($\tau$ is a leaf)
        $\mathbf{F}^{\tau}_{\text{c,ci}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{A}^{-1}_{\text{ci,ci}} \end{bmatrix}$   *[pot.] ← [body load]*
        $\mathbf{H}^{\tau}_{\text{ce,ci}} = \mathbf{D}_{\text{ce,c}}\mathbf{F}^{\tau}_{\text{c,ci}}$   *[deriv.] ← [body load]*
        $\mathbf{S}^{\tau}_{\text{c,ce}} = \begin{bmatrix} \mathbf{I} \\ -\mathbf{A}^{-1}_{\text{ci,ci}}\mathbf{A}_{\text{ci,ce}} \end{bmatrix}$   *[pot.] ← [pot.]*
        $\mathbf{T}^{\tau}_{\text{ce,ce}} = \mathbf{D}_{\text{ce,c}}\mathbf{S}_{\text{c,ce}}$   *[deriv.] ← [pot.] (NfD operator)*
    else
        Let $\alpha$ and $\beta$ be the children of $\tau$.
        Partition $I^{\alpha}_{\text{e}}$ and $I^{\beta}_{\text{e}}$ into vectors $I_1$, $I_2$, and $I_3$ as shown in Figure 3.2.
        $\mathbf{X}^{\tau}_{\text{ci,ci}} = \left(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\right)^{-1}$   *[pot.] ← [deriv.]*
        $\mathbf{S}^{\tau}_{\text{ci,ce}} = \mathbf{X}^{\tau}_{\text{ci,ci}}\left[-\mathbf{T}^{\alpha}_{3,1} \mid \mathbf{T}^{\beta}_{3,2}\right]$   *[pot.] ← [pot.]*
        $\mathbf{T}^{\tau}_{\text{ce,ce}} = \begin{bmatrix} \mathbf{T}^{\alpha}_{1,1} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \mathbf{S}^{\tau}_{\text{ci,ce}}$   *[deriv.] ← [pot.] (NfD operator).*
    end if
end for
```

Figure 4.2: Build stage.

Algorithm 2 (Solver for problems with body load)

This program constructs an approximation $\mathbf{u}$ to the solution u of (4.2). It assumes that all matrices required to represent the solution operator have already been constructed using Algorithm 3. It is assumed that if node τ is a parent of node σ, then $\tau < \sigma$.

Upwards pass — construct all particular solutions:

```
for $\tau = N_{\rm boxes},\, N_{\rm boxes}-1,\, N_{\rm boxes}-2,\, \dots,\, 1$
    if ($\tau$ is a leaf)
        # Compute the derivatives of the local particular solution.
        $\mathbf{h}^{\tau}_{\rm ce} = \mathbf{H}^{\tau}_{\rm ce,ci}\,\mathbf{g}^{\tau}_{\rm ci}$
    else
        Let $\alpha$ and $\beta$ denote the children of $\tau$.
        # Compute the local particular solution.
        $\mathbf{w}^{\tau}_{\rm ci} = \mathbf{X}^{\tau}_{\rm ci,ci}\,\big(-\mathbf{h}^{\alpha}_{3} + \mathbf{h}^{\beta}_{3}\big).$
        # Compute the derivatives of the local particular solution.
        $\mathbf{h}^{\tau}_{\rm ce} = \begin{bmatrix} \mathbf{h}^{\alpha}_{1} \\ \mathbf{h}^{\beta}_{2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \mathbf{w}^{\tau}_{\rm ci}.$
    end if
end for
```

Downwards pass — construct all potentials:

```
# Use the provided Dirichlet data to set the solution on the exterior of the root.
$\mathbf{u}(k) = f(\boldsymbol{y}_k)$ for all $k \in I^{1}_{\rm ge}$.
for $\tau = 1,\, 2,\, 3,\, \dots,\, N_{\rm boxes}$
    if ($\tau$ is a parent)
        # Add the homogeneous term and the particular term.
        $\mathbf{u}^{\tau}_{\rm ci} = \mathbf{S}^{\tau}_{\rm ci,ce}\,\mathbf{u}^{\tau}_{\rm ce} + \mathbf{w}^{\tau}_{\rm ci}.$
    else
        # Add the homogeneous term and the particular term.
        $\mathbf{u}^{\tau}_{\rm c} = \mathbf{S}^{\tau}_{\rm c,ce}\,\mathbf{u}^{\tau}_{\rm ce} + \mathbf{F}^{\tau}_{\rm c,ci}\,\mathbf{g}^{\tau}_{\rm ci}.$
    end
end for
```

Figure 4.3: Solve stage.

$$\begin{array}{c|cccccc}
0 & 0 & & & & & \\
2\gamma & \gamma & \gamma & & & & \\
c_3 & a_{3,1} & a_{3,2} & \gamma & & & \\
\vdots & \vdots & \vdots & \ddots & \ddots & & \\
c_{s-1} & a_{s-1,1} & a_{s-1,2} & a_{s-1,3} & \cdots & \gamma & \\
1 & b_1 & b_2 & b_3 & \cdots & b_{s-1} & \gamma \\
\hline
 & b_1 & b_2 & b_3 & \cdots & b_{s-1} & \gamma
\end{array}$$

ESDIRK methods offer the advantages of stiff accuracy and L-stability. They are particularly attractive when used in conjunction with direct solvers since the elliptic solve required in each stage involves the same coefficient matrix $(I - \Delta t \gamma \mathcal{L})$, where Δt is the time-step. The only significant downside to the HPS algorithm is the high memory cost, but in this case we will only need to store solution operators for $(I - \Delta t \gamma \mathcal{L})$.

In general we split the right hand side of (4.1) into a stiff part, $F^{[1]}$, that will be treated implicitly using ESDIRK methods, and a part, $F^{[2]}$, that will be treated explicitly (with a Butcher table denoted $\hat{c}$, $\hat{A}$, and $\hat{b}$). Precisely we will use the Additive Runge-Kutta (ARK) methods by Carpenter and Kennedy, [25], of order 3,4 and 5.

We may choose to formulate the Runge–Kutta method in terms of either solving for slopes or solving for stage solutions. We denote these the k_i formulation and the u_i formulation, respectively. When solving for slopes the stage computation is

$$k_i^n = F^{[1]}(t_n + c_i \Delta t, u^n + \Delta t \sum_{j=1}^{s} a_{ij} k_j^n + \Delta t \sum_{j=1}^{s} \hat{a}_{ij} l_j^n), \quad i = 1, \ldots, s, \tag{4.3}$$

$$l_i^n = F^{[2]}(t_n + c_i \Delta t, u^n + \Delta t \sum_{j=1}^{s} a_{ij} k_j^n + \Delta t \sum_{j=1}^{s} \hat{a}_{ij} l_j^n), \quad i = 1, \ldots, s. \tag{4.4}$$

Note that the explicit nature of (4.4) is encoded in the fact that the elements on the diagonal and above in $\hat{A}$ are zero. Once the slopes have been computed the solution at the next time-step is assembled as

$$u^{n+1} = u^n + \Delta t \sum_{j=1}^{s} b_j k_j^n + \Delta t \sum_{j=1}^{s} \hat{b}_j l_j^n. \tag{4.5}$$

If the method is instead formulated in terms of solving for the stage solutions the implicit solves take the form

$$u_i^n = u^n + \Delta t \sum_{j=1}^{s} \Big(a_{ij} F^{[1]}(t_n + c_j \Delta t, u_j^n) + \hat{a}_{ij} F^{[2]}(t_n + c_j \Delta t, u_j^n) \Big),$$

and the explicit update for u^{n+1} is given by

$$u^{n+1} = u^n + \Delta t \sum_{j=1}^{s} b_j (F^{[1]}(t_n + c_j \Delta t, u_j^n) + F^{[2]}(t_n + c_j \Delta t, u_j^n)).$$

The two formulations are algebraically equivalent but offer different advantages. For example, when working with the slopes we do not observe (see experiments presented below) any order reduction due to time-dependent boundary conditions (see e.g. the analysis by Rosales et al. [34]). On the other hand and as discussed in some detail below, in solving for the slopes the HPS framework requires an additional step to enforce continuity.

We note that it is generally preferred to solve for the slopes when implementing implicit Runge-Kutta methods, particularly when solving very stiff problems where the influence of roundoff (or solver tolerance) errors can be magnified by the Lipschitz constant when solving for the stages directly.

Remark 10 The HPS method for elliptic solves was previously used in [24], which considered a linear hyperbolic equation

$$\frac{\partial u}{\partial t} = \mathcal{L} u(\boldsymbol{x}, t), \quad \boldsymbol{x} \in \Omega, t > 0,$$

where $\mathcal{L}$ is a skew-Hermitian operator. The evolution of the numerical solution can be performed by approximating the propagator $\exp(\tau \mathcal{L}) : L^2(\Omega) \to L^2(\Omega)$ via a rational approximation

$$\exp(\tau \mathcal{L}) \approx \sum_{m=-M}^{M} b_m (\tau \mathcal{L} - \alpha_m)^{-1}.$$

If application of $(\tau \mathcal{L} - \alpha_m)^{-1}$ to the current solution can be reduced to the solution of an elliptic-type PDE it is straightforward to apply the HPS scheme to each term in the approximation. A drawback with this approach is that multiple operators must be formed and it is also slightly more convenient to time step non-linear equations using the Runge-Kutta methods we use here.

There are two modifications to the HPS algorithm that are necessitated by the use of ARK time integrators, we discuss these in the next two subsections.

4.3.1 Neumann Data Correction in the Slope Formulation

In the HPS algorithm the PDE is enforced on interior nodes and continuity of the normal derivative is enforced on the leaf boundary. Now, due to the structure of the update formula (4.5), if at some time u^n has a component in the null space of the operator that is used to solve for a slope k_i, then this will remain throughout the solution process. Although this does not affect the stability of the method it may result in loss of relative accuracy as the solution evolves. As a concrete example consider the heat equation

$$u_t = u_{xx}, \quad x \in [0,2], \quad t > 0, \tag{4.6}$$

with the initial data $u(x,0) = 1 - |x-1|$, and with homogenous Dirichlet boundary conditions. We discretize this on two leaves which we denote by α and β.

Now in the k_i formulation, we solve several PDEs for the k_i values and update the solution as

$$u^{n+1} = u^n + \Delta t \sum_{j=1}^{s} b_j k_j^n.$$

Since the individual slopes k_i have continuous derivatives across the boundary the kink in u^n will be propagated to u^{n+1}. The function $u(x,t) = 1 - |x-1|$ is in the null space of the operators B_i that solve $k_i = B_i u^n$ and in this particular example we would end up with the incorrect steady state solution $u(x,t) = 1 - |x-1|$.

Fortunately, this can easily be mitigated by adding a consistent penalization of the jump in the derivative of the solution during the merging of two leaves. Recall that originally we enforced continuity of the derivative across the boundary by setting v_3 equal in the DtN operators for each leaf.

Also recall that the subscripts 1, 2, and 3 denote the values on index vectors I_1, I_2, and I_3 for the exterior nodes of the children. In our Runge-Kutta method k_i denotes the solution at intermediate stage i, so now $k_{i,3}$ denotes the solution k_i at intermediate stage i on the index vector I_3. Now $h^{k,\alpha}$ and $h^{k,\beta}$ denote the spectral derivative on each child's boundary for the particular solution to the PDE for k_i and are already present in [2]. However, $h^{u,\alpha}$ and $h^{u,\beta}$, which denote the spectral derivative of u^n on the boundary from each child box, are new additions. For each leaf box we have the Dirichlet to Neumann equations

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}^{\alpha}_{1,1} & \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\alpha}_{3,1} & \mathbf{T}^{\alpha}_{3,3} \end{bmatrix} \begin{bmatrix} \mathbf{k}_{i,1} \\ \mathbf{k}_{i,3} \end{bmatrix} + \begin{bmatrix} \mathbf{h}^{k,\alpha}_{1} \\ \mathbf{h}^{k,\alpha}_{3} \end{bmatrix}, \tag{4.7}$$

$$\begin{bmatrix} \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}^{\beta}_{2,2} & \mathbf{T}^{\beta}_{2,3} \\ \mathbf{T}^{\beta}_{3,2} & \mathbf{T}^{\beta}_{3,3} \end{bmatrix} \begin{bmatrix} \mathbf{k}_{i,2} \\ \mathbf{k}_{i,3} \end{bmatrix} + \begin{bmatrix} \mathbf{h}^{k,\beta}_{2} \\ \mathbf{h}^{k,\beta}_{3} \end{bmatrix}. \tag{4.8}$$

Now we enforce that the difference in Neumann data $\mathbf{v}_3$ across the boundary in the first stage value k_1 must compensate for any difference in Neumann data for $\mathbf{u}$ across the boundary,

$$\left(\mathbf{T}^{\alpha}_{3,1}\,\mathbf{k}_{1,1} + \mathbf{T}^{\alpha}_{3,3}\,\mathbf{k}_{1,3} + \mathbf{h}^{k,\alpha}_{3}\right) - \left(\mathbf{T}^{\beta}_{3,2}\,\mathbf{k}_{1,2} + \mathbf{T}^{\beta}_{3,3}\,\mathbf{k}_{1,3} + \mathbf{h}^{k,\beta}_{3}\right) = -\frac{1}{b_1 \Delta t}(\mathbf{h}^{u,\alpha}_{3} - \mathbf{h}^{u,\beta}_{3}),$$

where b_1 is the first element of the vector b in the butcher diagram for the Runge-Kutta method. This gives

$$\mathbf{k}_{1,3} = \left(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\right)^{-1}\left(\mathbf{T}^{\beta}_{3,2}\mathbf{k}_{1,2} - \mathbf{T}^{\alpha}_{3,1}\mathbf{k}_{1,1} + \mathbf{h}^{k,\beta}_{3} - \mathbf{h}^{k,\alpha}_{3} - \frac{1}{b_1 \Delta t}(\mathbf{h}^{u,\alpha}_{3} - \mathbf{h}^{u,\beta}_{3})\right),$$

along with the modified equation for the fluxes of the particular solution on the parent box

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix} = \left(\begin{bmatrix} \mathbf{T}^{\alpha}_{1,1} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \left(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\right)^{-1}\left[-\mathbf{T}^{\alpha}_{3,1} \mid \mathbf{T}^{\beta}_{3,2}\right] \right) \begin{bmatrix} \mathbf{k}_{1,1} \\ \mathbf{k}_{1,2} \end{bmatrix} +$$
$$\begin{bmatrix} \mathbf{h}^{k,\alpha}_{1} \\ \mathbf{h}^{k,\beta}_{2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \left(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\right)^{-1}\left(\mathbf{h}^{\beta}_{3} - \mathbf{h}^{\alpha}_{3} - \frac{1}{b_1 \Delta t}(\mathbf{h}^{u,\alpha}_{3} - \mathbf{h}^{u,\beta}_{3})\right).$$

The above initial data is of course extreme but we note that the problem persists for any non-polynomial initial data with the size of the (stationary) error depending on resolution of the simulation. We further note that the described modification removes this problem without affecting the accuracy or performance of the overall algorithm. For example, consider the problem

$$\begin{cases} u_t = \Delta u(x,t), & \boldsymbol{x} \in [0,2], \\ u(x=0,t) = 0, & \\ u(x=2,t) = 0, & \\ u(x,t=0) = \exp(-200(x-1)^2). & \end{cases} \tag{4.9}$$

The previous problem had a discontinuity in the derivative of the initial condition across the boundary, but this initial condition has a continuous derivative. However, the spectral approximation to the derivative is discontinuous across the boundary. If we do not apply the correction to enforce continuity of the derivative we converge to a steady state solution which is linear on each leaf, thus satisfying $u_t = u_{xx} = 0$, but retains the same discontinuity in the spectral derivative. This is demonstrated in Figures 4.4 and 4.5, which show the initial condition and steady state solution with no correction to the Neumann data.

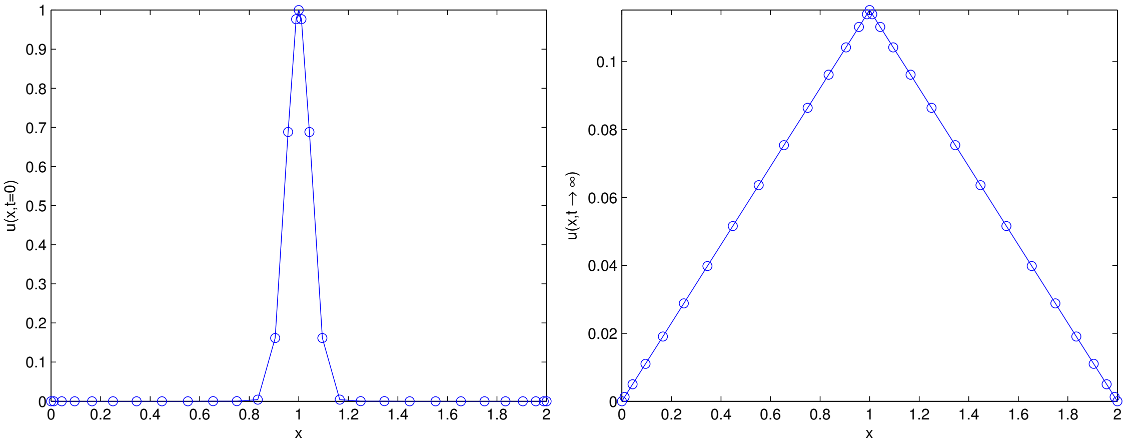
Remark 11 Although for linear constant coefficient PDE it may be possible to project the initial data in a way so that interior affine functions do not cause the difficulty above, for greater generality, we have chosen to enforce the extra penalization throughout the time evolution.

Remark 12 When utilizing the u_i formulation in a purely implicit problem we do not encounter the difficulty described above. This is because we enforce continuity of the derivative in u_s^n when solving

$$(I - \Delta t \gamma \mathcal{L}) u_s^n = u^n + \Delta t \mathcal{L}\Big(\sum_{j=1}^{s-1} a_{sj} u_j^n\Big) + \Delta t \sum_{j=1}^{s-1} a_{sj} g(x, t_n + c_j \Delta t),$$

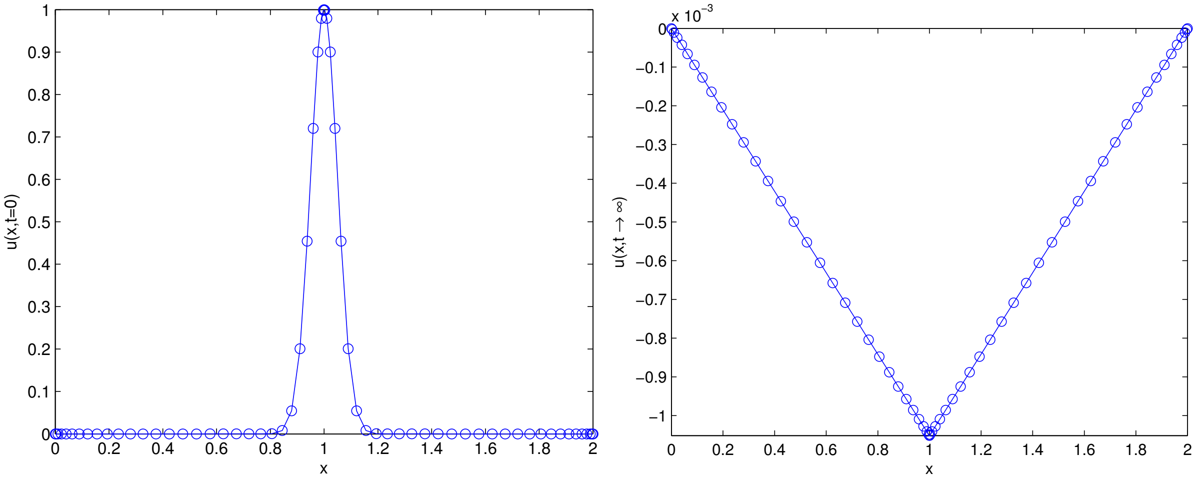
followed by the update $u^{n+1} = u_s^n$.

When utilizing the u_i formulation in an additive Runge-Kutta problem the details are a bit more complicated. We can still have a solution with discontinuous Neumann data in u^{n+1}, but this is actually related to Section 4.3.2 and will be discussed there.



(a) (b)

Figure 4.4: (a) The initial condition given in (4.9) with $p = 16$ Chebyshev nodes per leaf. The spectral derivative at $x = 1$ is found to be $u_x = .115$ from the left leaf and $u_x = -.115$ from the right leaf. (b) The steady state solution found by using ESDIRK in time with the HPS scheme in space to evolve the PDE without correction for the difference in derivative. We see the solution converges to a linear function on each leaf so that $u_{xx} = 0$ on each leaf, but the difference in derivative across the boundary is retained. The error in the spectral derivative is not as severe in this example, but it is still significant that the solution does not decay to zero.



(a) (b)

Figure 4.5: (a) The initial condition given in (4.9) with $p = 32$ Chebyshev nodes per leaf. The spectral derivative at $x = 1$ is found to be $u_x = -1.05 \times 10^{-3}$ from the left leaf and $u_x = 1.05 \times 10^{-3}$ from the right leaf. (b) The steady state solution found by using ESDIRK in time with the HPS scheme in space to evolve the PDE without correction for the difference in derivative. We see the solution converges to a linear function on each leaf so that $u_{xx} = 0$ on each leaf, but the difference in derivative across the boundary is retained.

4.3.2 Enforcing Continuity in the Explicit Stage

The second modification is to the first explicit stage in the k_i formulation. Solving a problem with no forcing this stage is simply

$$k_1^n = \mathcal{L}(u_n).$$

When, for example, $\mathcal{L}$ is the Laplacian, we must evaluate it on all nodes on the interior of the physical domain. This includes the nodes on the boundary between two leafs where the spectral approximation to the Laplacian can be different if we use values from different leaves. The seemingly obvious choice, replacing the Laplacian on the leaf boundary by the average, leads to instability. However, stability can be restored if we enforce $k_1^n = \mathcal{L}(u_n)$ on the interior of each leaf and continuity of the derivative across each leaf boundary. Algorithmically, this is straightforward as these are the same conditions that are enforced in the regular HPS algorithm, except in this case we simply have an identity equation for k_1 on the interior nodes instead of a full PDE.

Although it is convenient to enforce continuity of the derivative using the regular HPS algorithm it can be done in a more efficient fashion by forming a separate system of equations involving only data on the leaf boundary nodes. In a single dimension on a discretization with n leafs this reduces the work associated with enforcing continuity of the derivative across leaf boundary nodes from solving $n \times (p-1) - 1$ equations for $n \times (p-1) - 1$ unknowns to solving a tridiagonal system of equations $n - 1$ equations for $n - 1$ unknowns.

In two dimensions the system is slightly different, but if we have $n \times n$ leafs with $p \times p$ Chebyshev nodes on each leaf then eliminating the explicit equations for the interior nodes reduces the system to $(p-2) \times 2n$ independent tridiagonal systems of $n-1$ equations with $n-1$ unknowns for a total of $(p-2) \times 2n \times (n-1)$ equations with $(p-2) \times 2n \times (n-1)$ unknowns.

When the u_i formulation is used for a fully implicit problem the intermediate stage values still require us to evaluate $\mathcal{L}u^n$, but this quantity only enters through the body load in the intermediate stage PDEs. The explicit first stage in this formulation is simply $u_1^n = u^n$. Furthermore, while we must calculate

$$u^{n+1} = u^n + \Delta t \mathcal{L}\Big(\sum_{j=1}^{s} a_{sj} u_j^n\Big),$$

this is equivalent to u_s^n since $b_j = a_{sj}$ and we simply take $u^{n+1} = u_s^n$.

When both explicit and implicit terms are present, we proceed differently. Now, the values of u_i^n look almost identical to the implicit case and we still avoid the problem of an explicit "solve" in u_1^n, but we also have

$$u^{n+1} = u^n + \Delta t \sum_{j=1}^{s} b_j (F^{[1]}(t_n + c_j \Delta t, u_j^n) + F^{[2]}(t_n + c_j \Delta t, u_j^n)).$$

The ESDIRK method has the property that $b_j = a_{sj}$, but for the explicit Runge-Kutta method we have $b_j \neq \hat{a}_{sj}$. When the explicit operator $F^{[2]}$ does not contain partial derivatives we need not enforce continuity of the derivative and can simply reformulate the method as

$$\begin{aligned} u^{n+1} &= u^n + \Delta t \sum_{j=1}^{s} a_{sj} (F^{[1]}(t_n + c_j \Delta t, u_j^n) + F^{[2]}(t_n + c_j \Delta t, u_j^n)) \\ &= u^n + \Delta t \sum_{j=1}^{s} \Big(a_{sj} F^{[1]}(t_n + c_j \Delta t, u_j^n) + \hat{a}_{sj} F^{[2]}(t_n + c_j \Delta t, u_j^n) \Big) \\ &\quad + \Delta t \sum_{j=1}^{s} (a_{sj} - \hat{a}_{sj}) F^{[2]}(t_n + c_j \Delta t, u_j^n) \\ &= u_s^n + \Delta t \sum_{j=1}^{s} (a_{sj} - \hat{a}_{sj}) F^{[2]}(t_n + c_j \Delta t, u_j^n). \end{aligned}$$

4.4 Boundary Conditions

The above description for Runge-Kutta methods does not address how to impose boundary conditions for a system of ODEs resulting from a discretization of a PDE. In particular, the different formulations incorporate boundary conditions in slightly different ways.

In this work we consider Dirichlet, Neumann, and periodic boundary conditions. For periodic boundary conditions the intermediate stage boundary conditions are enforced to be periodic for both formulations. As the k_i stage values are approximations to the time derivative of u, the imposed Dirichlet boundary conditions for $x \in \Gamma$ are $k_i^n = u_t(x, t_n + c_i \Delta t)$. When solving for u_i one may attempt to enforce boundary conditions using $u_i = u(x, t + c_i \Delta t), x \in \Gamma$. However, as demonstrated in part two of this series and discussed in detail in [34], this results in order reduction for time dependent boundary conditions.

In the HPS algorithm, Neumann or Robin boundary conditions are mapped to Dirichlet boundary conditions using the linear Dirichlet to Neumann operator as discussed for example in [2].

4.5 Time Dependent Boundary Conditions

This section discusses time-dependent boundary conditions within the two different Runge-Kutta formulations. In particular, we investigate the order reduction that has been documented in [34] for implicit Runge-Kutta methods and earlier in [6] for explicit Runge-Kutta methods.

In this first experiment, introduced in [34], we solve the heat equation in one dimension

$$u_t = u_{xx} + f(t), \qquad x \in [0, 2], \quad t > 0. \tag{4.10}$$

We set the initial data, Dirichlet boundary conditions and the forcing $f(t)$ so that exact solution is $u(x, t) = \cos(t)$. This example is designed to eliminate the effect of the spatial discretization, with the solution being constant in space and allows for the study of possible order reduction near the boundaries.

We use the HPS scheme in space and use 32 leafs with $p = 32$ Chebyshev nodes per leaf. We apply the third, fourth, and fifth order ESDIRK methods from [25]. We consider solving for the intermediate solutions, or as we refer to it below "the u_i formulation" with the boundary condition enforced as $u_i^n = \cos(t_n + c_i \Delta t)$. We also consider solving for the stages, which we refer to as "the k_i formulation" with boundary conditions imposed as $k_i^n = -\sin(x, t_n + c_i \Delta t)$.

Error reduction for time dependent boundary conditions has been studied both in the context of explicit Runge-Kutta methods in e.g. [6] and more recently for implicit Runge-Kutta methods in [34]. In [34] the authors report observed orders of accuracy equal to two (for the solution u)

for DIRK methods of order 2, 3, and 4 for the problem (4.10) discretized with a finite difference method on a fine grid (the spatial errors are zero) using the u_i formulation.

Figures 4.6 and 4.7 show the error for the third and fifth order ESDIRK methods, respectively, as a function of x for a single step and at the final time $t = 1$. Figure 4.8 shows the maximum error for the third, fourth, and fifth order methods as a function of time step Δt after a single step and at the final time $t = 1$.

In general, for a method of order p we expect that the single step error decreases as Δt^{p+1} while the global error decreases as Δt^p. However, with time dependent boundary conditions implemented as $u_i^n = \cos(t_n + c_i\Delta t)$ the results in [34] indicate that the rate of convergence will not exceed two for the single step or global error.

The results for the third order method ($p = 3$) displayed in Figure 4.6 show that the single step error decreases as Δt^{p+1} while the global error decreases as Δt^p, which is better than the results documented in [34]. However, we still see that a boundary layer appears to be forming, but it is of the same order as the error away from the boundary. The results for the fifth order method ($p = 5$) displayed in Figure 4.7 show that the single step error decreases as Δt^4 while the global error decreases as Δt^3, which is still better than the results documented in [34]. However, the boundary layer is giving order reduction from Δt^{p+1} for the single step error and Δt^p for the global error. We note that our observations differ from those in [34] but that this possibly can be attributed to the use of a ESDIRK method rather than a DIRK method.

We repeat the experiment but now we use the k_i formulation for Runge-Kutta methods and for the boundary condition we enforce $k_i^n = -\sin(t_n + c_i\Delta t)$. The intuition here is that k_i^n is an approximation to u_t at time $t_n + c_i\Delta t$ and we use the value of u_t for the boundary condition of k_i^n. Intuitively we expect that the fact that we reduce the index of the system of differential algebraic equation in the u_i formulation by differentiating the boundary conditions can restore the design order of accuracy.

In the previous examples the Runge-Kutta method introduced an error on the interior while the solution on the boundary was exact. If the error on the boundary is on the same order of magnitude as the error on the interior then the error in u_{xx} is of the correct order, but when the value of u is exact on the boundary it introduces a larger error in u_{xx}. In the k_i formulation, for each intermediate stage we find $u_{xx} = 0$ and then $k_i^n = -\sin(t_n + c_i\Delta t)$ on the interior and on the boundary. So at a fixed time the solution is constant in x and a boundary layer does not form. Additionally, the error is constant in x at any fixed time and for a method of order p we obtain the expected behavior where the single step error decreases as Δt^{p+1} and the global error decreases as Δt^p.

Figure 4.8 shows the maximum error for the third, fourth, and fifth order methods as a function of time step Δt after a single step and at the final time $t = 1$. The results show that the methods behave exactly as we expect. The single step error behaves as Δt^{p+1} for the 3rd and 5th order methods and Δt^{p+2} for the 4th order method. The 4th order method gives 6th order error in a single step because the exact solution is $u(x,t) = \cos(t)$, which has every other derivative equal to zero at $t = 0$ and for a single step we start at $t = 0$. The global error behaves as Δt^p for each method.

4.6 Numerical Investigation of Stability

The ESDIRK methods we use here are known to be L-stable, [25], and to confirm that we still have unconditional stability with the HPS algorithm we compute the matrix Q that propagates the heat equation on the domain $0 \leq x \leq 2$ and with homogenous Dirichlet boundary conditions for one step. That is given a vector $\mathbf{v}^n$ we have that $\mathbf{v}^{n+1} = Q\mathbf{v}^n$. In order for the fully discrete scheme to be stable the spectral radius of the matrix Q should not exceed one. To check this we form Q for various ratios of $\Delta t/\Delta x$ and for $p = 4, 8, 16, 32, 64$, for both the u_i formulation and the k_i formulation and for ESDIRK3 and ESDIRK4. The spectral radii of Q for the different combinations are plotted in Figure 4.9. The predictions for the u_i method (dashed lines) clearly show that the method is stable. However, based on the computed spectral radii for the k_i method, it would appear as if the k_i method is unstable. Fortunately this is not the case. For the cases where the spectral radii of Q indicates instability we tested the accuracy of the eigenvalue computation by evolving random initial data. In all cases the random initial data decays towards zero very rapidly and we could never observe any unbounded growth

To briefly explain why the two methods produce an evolution operator with a different spectral radius we consider an extreme case with $\Delta t = 10^8$, 4 leafs, $p = 64$, and the fifth order ESDIRK method. When computing the evolution operator Q we see exactly $N_{pan} - 1$ eigenvalues all greater than or equal to 600 and all other eigenvalues smaller than 4×10^{-8}. With a very large time step we would expect all eigenvalues to be extremely small. We plot the $N_{pan} - 1$ eigenvectors associated with the $N_{pan} - 1$ eigenvalues and see them in Figure 4.10.

We see the $N_{pan} - 1$ eigenvectors are all piecewise linear with discontinuous Neumann data across each leaf boundary. These eigenvectors span the space of all piecewise linear functions on the domain. This is counter-intuitive because the Neumann data correction discussed in Section 4.3.1 should smooth the derivative across the leaf interface, meaning we would not find $\mathbf{v}^{n+1} = Q\mathbf{v}^n = \lambda\mathbf{v}^n$. Furthermore, when running the HPS algorithm on any of these eigenvectors the eigenvectors decay as expected and the HPS algorithm smooths out the derivative across the interface.

Upon investigation, we look at the equation

$$\mathbf{k}_{1,3} = \left(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\right)^{-1}\left(\mathbf{T}^{\beta}_{3,2}\mathbf{k}_{1,2} - \mathbf{T}^{\alpha}_{3,1}\mathbf{k}_{1,1} + \mathbf{h}^{k,\beta}_{3} - \mathbf{h}^{k,\alpha}_{3} - \frac{1}{b_1\Delta t}(\mathbf{h}^{u,\alpha}_{3} - \mathbf{h}^{u,\beta}_{3})\right),$$

which enforces a sufficient jump in derivative for k_1^n so that u^{n+1} has a continuous derivative. When applying this correction to any of the eigenvectors the term $\mathbf{h}^{k,\beta}_{3} - \mathbf{h}^{k,\alpha}_{3}$ is roughly of the same order as $\frac{1}{\Delta t}(\mathbf{h}^{u,\alpha}_{3} - \mathbf{h}^{u,\beta}_{3})$ and the jump in derivative is properly corrected. However, when we apply the HPS algorithm to a unit basis vector e_i we find the two terms are of significantly different orders of magnitude. In the extreme case of $p = 64$, $N_{pan} = 10$, $k = 10^8$, and the fifth order ESDIRK method we find the jump in derivative for the initial data e_{63} is on the order of 10^4, but the term that corrects the jump in Neumann data is 15 orders of magnitude smaller than the first term and the correction to the Neumann data is effectively not performed for this initial data and the jump in Neumann data grows after the first time step and then it is corrected after the second time step. In practice for any reasonably smooth data the two methods perform the same, but the

k formulation breaks down when applying the method to functions where the spectral derivatives have large differences across the leaf boundaries.

We always see $N_{pan} - 1$ eigenvalues significantly greater than the others. Figure 4.10 shows this for the case of $\Delta t = 10^8$, 16 leafs, $p = 64$, and the fifth order ESDIRK method.

Remark 13 The essential argument is that both methods are unconditionally stable and perform the same as long as our initial data does not have large jumps in the derivative across the boundary. As long as the initial data is sufficiently resolved in space both methods performs well. However, applying the algorithm to the same initial data gives slightly different results when the initial data is not sufficiently resolved. The equation we use to correct for the jump in derivative is

$$\mathbf{k}_{1,3} = \left(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\right)^{-1}\left(\mathbf{T}^{\beta}_{3,2}\mathbf{k}_{1,2} - \mathbf{T}^{\alpha}_{3,1}\mathbf{k}_{1,1} + \mathbf{h}^{k,\beta}_{3} - \mathbf{h}^{k,\alpha}_{3} - \frac{1}{b_1\Delta t}(\mathbf{h}^{u,\alpha}_{3} - \mathbf{h}^{u,\beta}_{3})\right),$$

which we will refer to as correction A. Earlier implementations of the algorithm spread the correction term out over the stage values by calculating

$$\mathbf{k}_{i,3} = \left(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\right)^{-1}\left(\mathbf{T}^{\beta}_{3,2}\mathbf{k}_{1,2} - \mathbf{T}^{\alpha}_{3,1}\mathbf{k}_{1,1} + \mathbf{h}^{k,\beta}_{3} - \mathbf{h}^{k,\alpha}_{3} - \frac{1}{\Delta t}(\mathbf{h}^{u,\alpha}_{3} - \mathbf{h}^{u,\beta}_{3})\right),$$

for each stage. This alternate correction B smoothed out the initial data after the first time step and was accurate for known analytical solutions. Although the spectral analysis gives incorrect predictions of the stability for both corrections, the different corrections give different results for under resolved solutions and correction A consistently gives better results. For both corrections the rate of convergence is the same.

4.7 Schrödinger Equation

Next we consider the Schrödinger equation for $u = u(x, y, t)$

$$\begin{aligned} i\hbar u_t &= -\frac{\hbar^2}{2M}\Delta u + V(x,y)u, \quad t > 0, \quad (x,y) \in [x_l, x_r] \times [y_b, y_t], \\ u(x,y,0) &= u_0(x,y). \end{aligned} \tag{4.11}$$

Here we nondimensionalize in a way equivalent to setting $M = 1, \hbar = 1$ in the above equation. We choose the potential to be the the harmonic potential

$$V(x, y) = \frac{1}{2}\left(x^2 + y^2\right).$$

This leads to an exact solution

$$u(x, y, t) = Ae^{-it}e^{-\frac{(x^2+y^2)}{2}}, \tag{4.12}$$

where we set $A = 1/\sqrt{\sqrt{\pi}}$ and solve until $t = 2\pi$ on the domain $(x, y) \in [-8, 8]^2$.

The computational domain is subdivided into $n_x \times n_y$ panels with $p \times p$ points on each panel. To begin, we study the order of accuracy with respect to leaf size. To eliminate the effect of time-stepping errors we scale $\Delta t = h^{p/q_{\mathrm{RK}}}$, where q_{RK} is the order of the Runge-Kutta method. In Figure 4.11 we display the errors as a function of the leaf size for $p = 4, 6, 8, 10, 12, 16$ and for the third and fifth order Runge-Kutta methods ($q_{RK} = 3, 5$). The rates of convergence are found for all three Runge-Kutta methods and summarized in Table 4.1. As can be seen from the table, $p = 4$ appears to converge at second order, while for higher p we generally observe a rate of convergence approaching to p.

Table 4.1: Estimated rates of convergence for different Runge Kutta methods and different orders of approximation.

p	4	6	8	10	12
ESDIRK3	2.59	5.73	7.72	9.69	11.47
ESDIRK4	1.89	6.47	7.82	9.76	11.69
ESDIRK5	1.84	4.42	7.69	9.71	11.48

In this problem the efficiency of the method is limited by the order of the Runge-Kutta methods. However, as our methods are unconditionally stable we may enhance the efficiency by using Richardson extrapolation to achieve a highly accurate solution in time. We solve the same problem, but now we fix $p = 12$ and take $5 \cdot 2^n$ time steps, with $n = 0, 1, \ldots, 5$. For the third order ESDIRK method we use 60×60 leaf boxes. For the fourth order ESDIRK method we use 90×90

leaf boxes. For the fifth order ESDIRK method we use 120 × 120 leaf boxes. Table 4.2 shows that we can easily achieve much higher accuracy by using Richardson extrapolation.

Table 4.2: Estimated errors at the final time after Richardson extrapolation.

q_{RK} / Extrapolations	0	1	2	3	4	5
3	1.32(-1)	1.01(-2)	1.27(-4)	1.17(-5)	6.98(-8)	8.62(-10)
4	2.70(-4)	6.46(-6)	1.23(-7)	2.95(-10)	1.59(-11)	3.70(-14)
5	1.28(-3)	9.67(-6)	6.30(-8)	1.86(-10)	4.11(-13)	9.27(-14)

Table 4.3: Errors computed against a p and h refined solution. The errors are maximum errors at the final time $t = 4$.

p / Panels	2	4	8	16	32
8	1.11(0)	1.39(-1)	8.74(-3)	1.50(-4)	2.45(-6)
rate	$*$	3.00	3.99	5.87	5.92
10	5.87(-1)	3.16(-2)	4.62(-4)	6.17(-6)	5.21(-8)
rate	$*$	4.21	6.10	6.22	6.89

Finally, we solve a problem without an analytic solution. In this problem the initial data

$$u(x, y, t) = 3\sin(x)\sin(y)e^{-(x^2+y^2)},$$

interacts with the weak and slightly non-symmetric potential

$$V(x, y) = 1 - e^{-(x+0.9y)^4},$$

allowing the solution to reach the boundary where we impose homogenous Dirichlet conditions.

We evolve the solution until time $t = 4$ using $p = 8$ and 10 and $2, 4, 8, 16$ and 32 leaf boxes in each direction of a domain of size 12 × 12. The errors computed against a reference solution with $p = 12$ and with 32 leaf boxes can be found in Table 4.3.

In Figure 4.12 we display snapshots of the magnitude of the solution at the initial time $t = 0$, the intermediate times $t \approx 1.07$, $t \approx 1.68$ and at the final time $t = 4.0$.

4.8 Burgers' Equation in Two Dimensions

As a first step towards a full blown flow solver we solve Burgers' equation in two dimensions using the additive Runge-Kutta methods described in the first part of this paper. Precisely, we solve the system

$$\mathbf{u}_t + \mathbf{u}\cdot\nabla\mathbf{u} = \varepsilon\Delta\mathbf{u}, \qquad \mathbf{x}\in[-\pi,\pi]^2, \quad t>0, \tag{4.13}$$

where $\mathbf{u} = [u(x,y,t), v(x,y,t)]^T$ is the vector containing the velocities in the x and y directions.

The first problem we solve uses the initial condition $\mathbf{u} = 5[-y,x]^T\exp(-3r^2)$ and the boundary conditions are taken to be no-slip boundary conditions on all sides. We solve the problem using 24×24 leafs, $p = 24$, $\varepsilon = 0.005$, and the fifth order ARK method found in [25]. We use a time step of $k = 1/80$ and solve until time $t_{\max} = 5$. The low viscosity combined with the initial condition produces a rotating flow resembling a vortex that steepens up over time.

In Figure 4.13 we can see the velocities at times $t = 0.5$ and $t = 1$. The fluid rotates and expands out and eventually forms a shock like transition. This creates a sharp flow region with large gradients resulting in a flow that may be difficult to resolve with a low order accurate method. These sharp gradients can be seen in the two vorticity plots in Figure 4.13 along with the speed and vorticity plots in Figure 4.14.

In our second experiment we consider a cross stream of orthogonal flows. We use an initial condition of

$$\mathbf{u} = [8y\, e^{-36\left(\frac{y}{2}\right)^8}, -8xe^{-36\left(\frac{x}{2}\right)^8}]^T, \tag{4.14}$$

and time independent boundary conditions that are compatible with the initial data.

This initial horizontal velocity drops to zero quickly as we approach $|y| = 0.5$. For $|y| < 0.5$ the exponential term approaches $\exp(0)$ and the velocity behaves like $u = 8y$. The flow has changed slightly by $t = 0.06$, but we can see in Figure 4.14 the flow is moving to the right for $y > 0$ and the flow is moving the left for $y < 0$ and all significant behavior is in $|y| < 0.5$. A plot of the velocity v would show similar behavior. We also use 24×24 leafs, $p = 24$, $\epsilon = 0.025$, $k = 1/200$, and $t_{\max} = 0.75$. We show plots of the horizontal velocity u and the dilatation at time $t = 0.06$

and $t = 0.15$. We only show plots before time $t = 0.15$ when the fluid is hardest to resolve and we observe that after $t = 0.15$ the cross streams begin to dissipate. This problem contains sharp interfaces inside $\mathbf{x} \in [-0.5, 0.5]^2$.

4.9 Convection-Diffusion

In this section we solve a convection-diffusion problem,

$$u_t = \kappa \Delta u - \alpha u_x, \tag{4.15}$$

where the diffusion term is very small in comparison to the convection term. We take $\kappa = 1$ and $\alpha = 150$. We solve the problem on a rectangle with eight holes. This domain and a sample coarse discretization showing the boundary of each leaf box can be seen in Figure 4.15. In the experiment we present we use a timestep $\Delta t = 0.001$, and on each leaf we use $p = 14$. The base grid uses $h = 1/8$, which is the leaf box width away for non-refined leaf boxes, and $N_{\text{ref}} = 5$ levels of refinement. This corresponds to roughly 970,000 discretization nodes. The time discretization uses the fifth order ESDIRK method.

We enforce Neumann boundary conditions on the north and south edges and all hole boundaries and periodic boundary conditions on the east and west boundaries. The initial condition is a sum of functions with compact support in a radius 2/5 around a center $\mathbf{r}_i$

$$u(x, 0) = \sum_{i=1}^{4} \mathbf{a}_i \exp\left(\frac{-1}{1 - (5|\mathbf{x} - \mathbf{r}_i|/2)^2}\right). \tag{4.16}$$

Here the amplitudes a_i are $\mathbf{a} = [1000, 300, 500, 100]$ and the centers are $\mathbf{r}_1 = [0.5, 1.45]$, $\mathbf{r}_2 = [1.3, 2.5]$, $\mathbf{r}_3 = [0.5, 5.3]$, and $\mathbf{r}_4 = [0.5, 4]$.

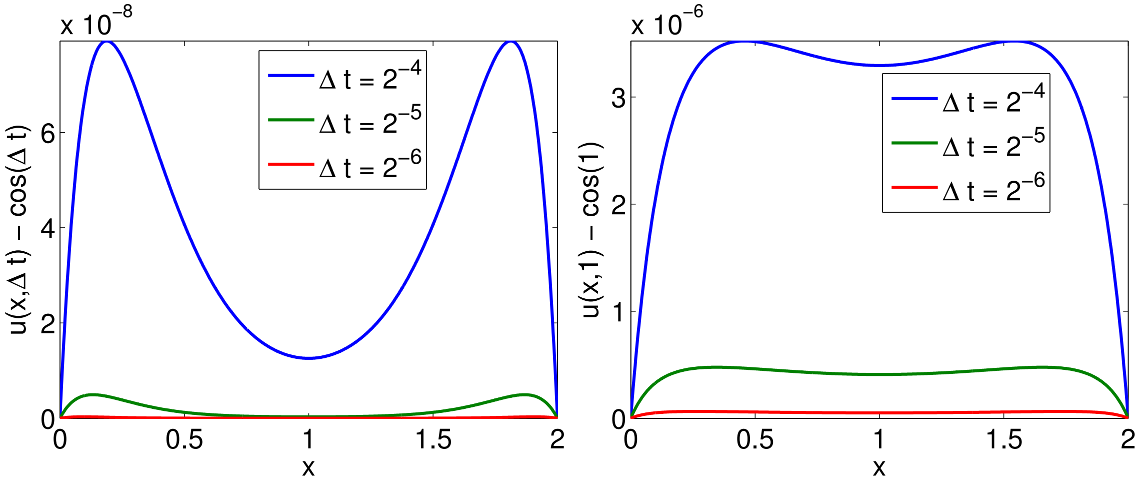
The initial condition can be seen in Figure 4.15 and in Figure 4.16 we display a sequence of snapshots. This problems illustrates the ability of the HPS scheme to handle problems with a large Peclet number (the ratio between advection and diffusion), a regime that may be difficult for iterative solvers.

4.10 Conclusion

In this two part series we have demonstrated that the spectrally accurate Hierarchial Poincaré-Steklov solver can be easily extended to handle time dependent PDE problems with a parabolic principal part by using ESDIRK methods. We have outlined the advantages of the two possible ways to formulate implicit Runge-Kutta methods within the HPS scheme and demonstrated the capabilities on both linear and non-linear examples.

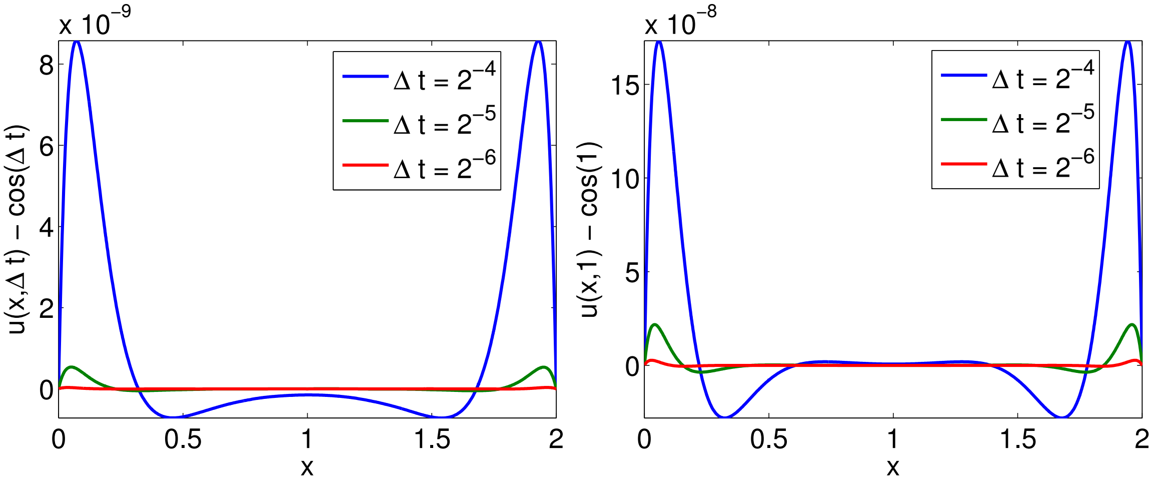
There are many avenues for future work, for example:

- Extension of the solvers to compressible and incompressible flows.
- Application of the current solvers to inverse and optimal design problems, in particular for problems where changes in parameters do not require new factorizations.



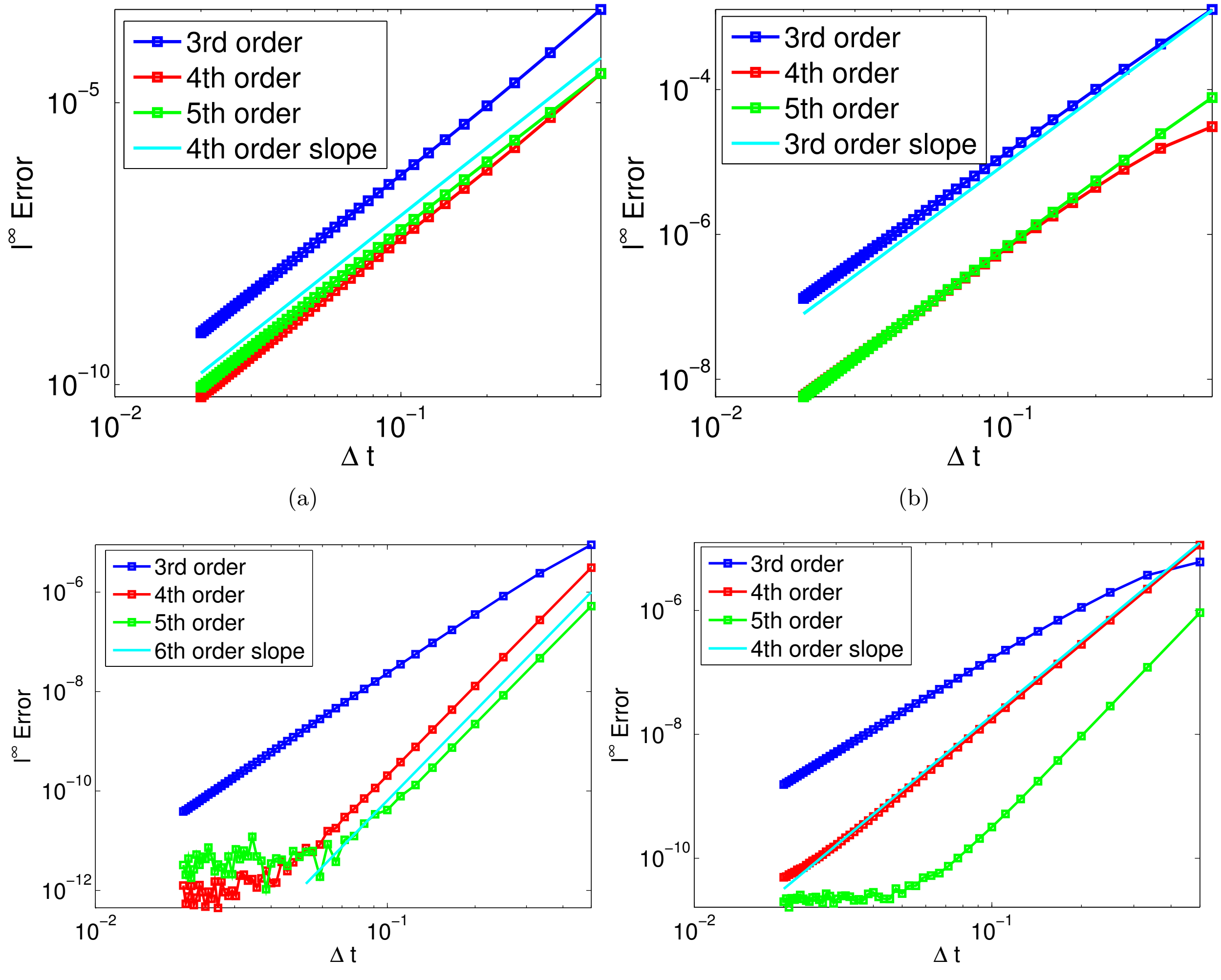
(a) (b)

Figure 4.6: The error in solving (4.10). Results are for a 3rd order ESDIRK. Figure (a) displays the single step error which converges with 4th order of accuracy. Figure (b) displays the global error at $t = 1$ converging at 3rd order. Both errors converge at one order higher than what is expected from the analysis in [34].



(a) (b)

Figure 4.7: The error in solving (4.10). Results are for the 5th order ESDIRK method. Figure (a) displays the single step error which converges with 4th order of accuracy. Figure (b) displays the global error at $t = 1$ converging at 3rd order. Both errors converge at one order higher than what is expected from the analysis in [34] but still lower than expected.



(a)

(b)

(c)

(d)

Figure 4.8: The error in solving (4.10) for the 3rd, 4th, and 5th order ESDIRK methods for a sequence of decreasing time steps. Figures (a) and (c) are errors after one time step and (b) and (d) are the errors at time $t = 1$. The top row are for the u_i formulation and the bottom row is for the k_i formulation. Note that the k_i formulation is free of order reduction.

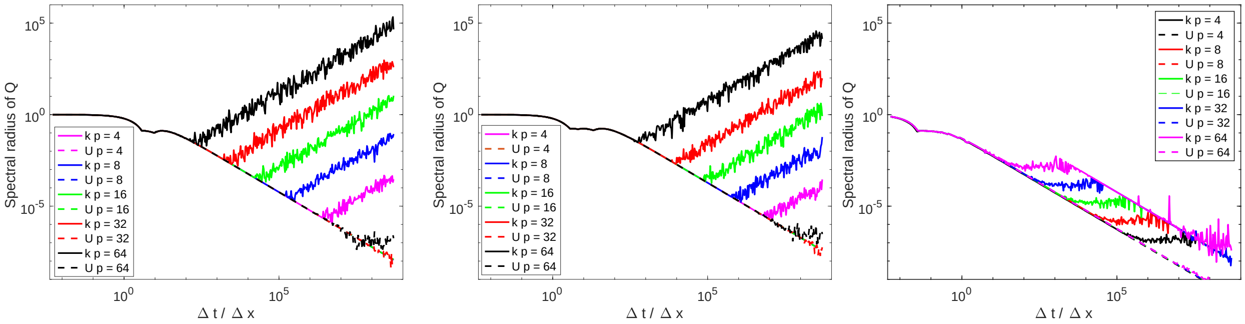


Figure 4.9: The spectral radius for the time-stepping matrix Q mapping $\mathbf{v}^{n+1} = Q\mathbf{v}^n$. The dashed lines are for the u_i formulation and the solid lines are for the k_i formulation. To the left are the results for the third order accurate method and to the middle are the results for the fourth order accurate method. In the right figure we have removed the $N_{pan} - 1$ largest eigenvalues and plotted the next largest for the third order method. For reasonable values of $\Delta t/\Delta x$ and p we see that the spectral radius is the same in each case. For extreme values of $\Delta t/\Delta x$ and p we see that only N_{pan} eigenvalues are larger than the largest eigenvalue for the U formulation and the next eigenvalue is the same as the largest eigenvalue for the U formulation.

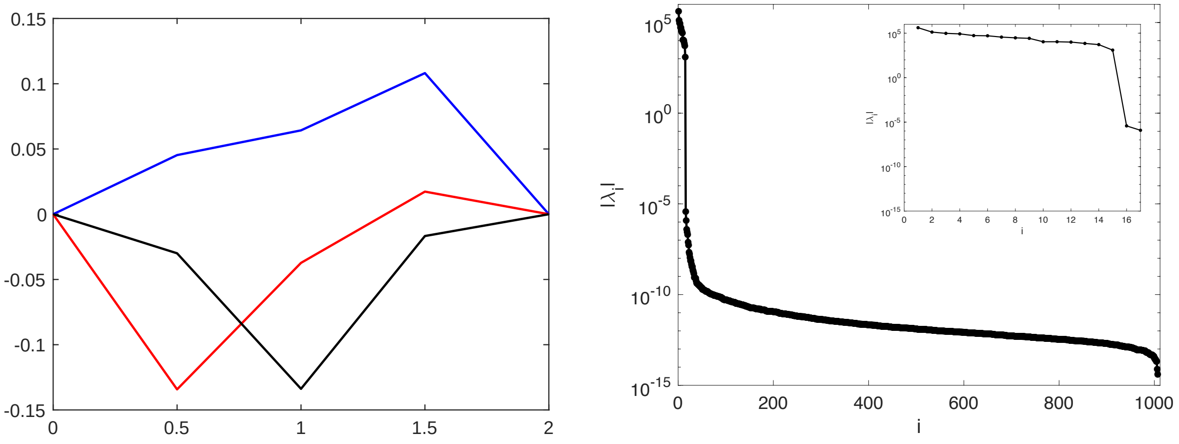


Figure 4.10: The left figure shows the piecewise linear eigenvectors associated with the $N_{pan} - 1$ largest eigenvalues with $N_{pan} = 4$. The right figure shows the distribution of $|\lambda_i|$ for $N_{pan} = 16$. We find $|\lambda_{15}| > 10^3$ and $|\lambda_{16}| < 10^{-5}$.

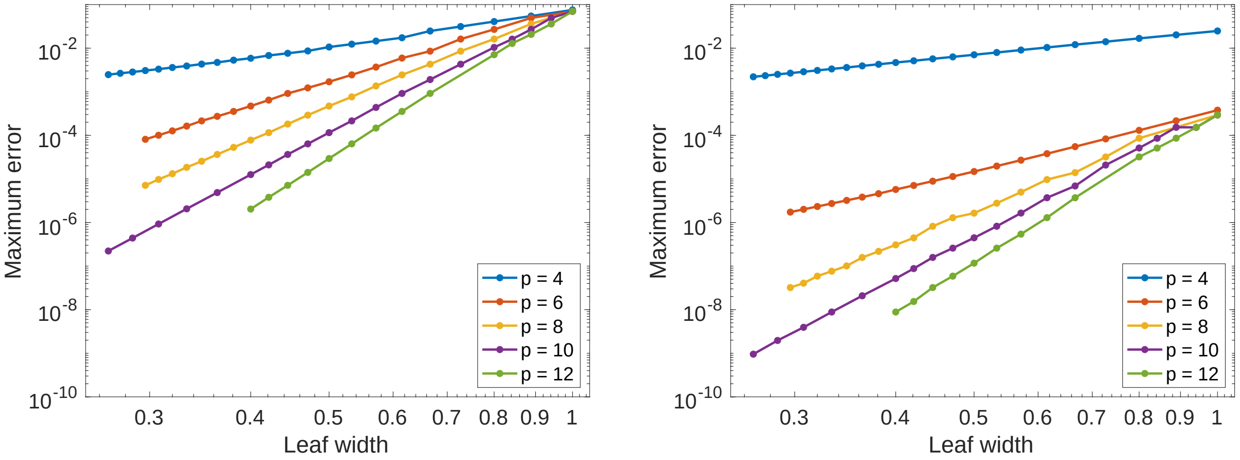
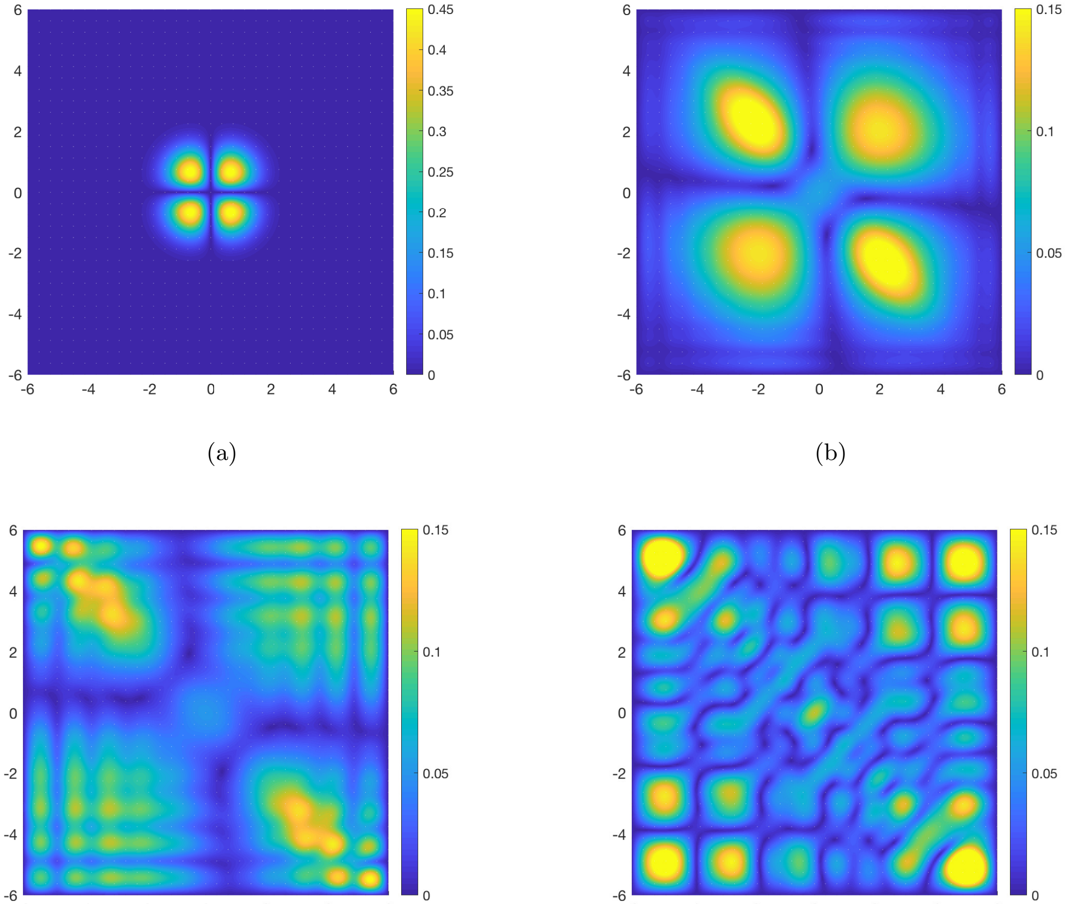


Figure 4.11: Error in the Schrödinger equation as a function of leaf size. The exact solution is (4.12).



(a)

(b)

(c)

(d)

Figure 4.12: Snapshots of the magnitude of the solution at the initial time $t = 0$, the intermediate times $t \approx 1.07$, $t \approx 1.68$ and at the final time $t = 4.0$.

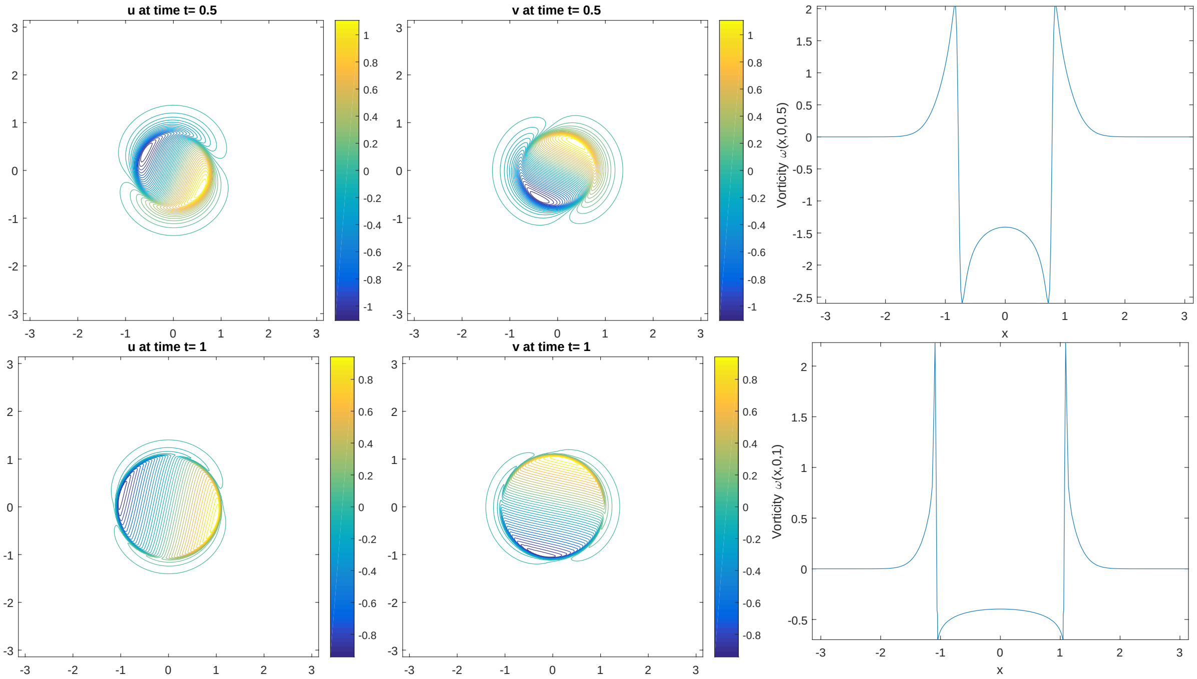


Figure 4.13: The four plots on the left show the velocities in the x and y directions at times $t = .5$ and $t = 1$. We can see the fluid rotating and expanding. The two plots in the third column show the vorticity at these times. We see sharp gradients near the edge of the rotating fluid.

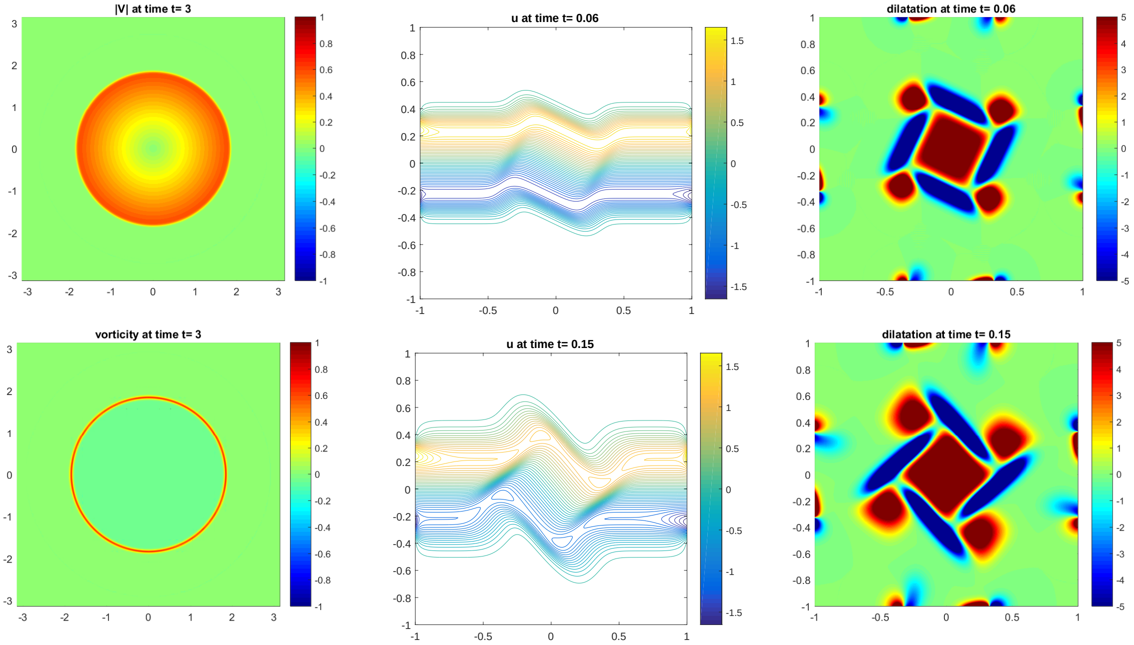


Figure 4.14: The plots in the first column correspond to the rotating shock problem. The second and third columns show the velocity in the x direction and the dilatation at times $t = 0.06$ and $t = 0.15$ for the cross flow problem.

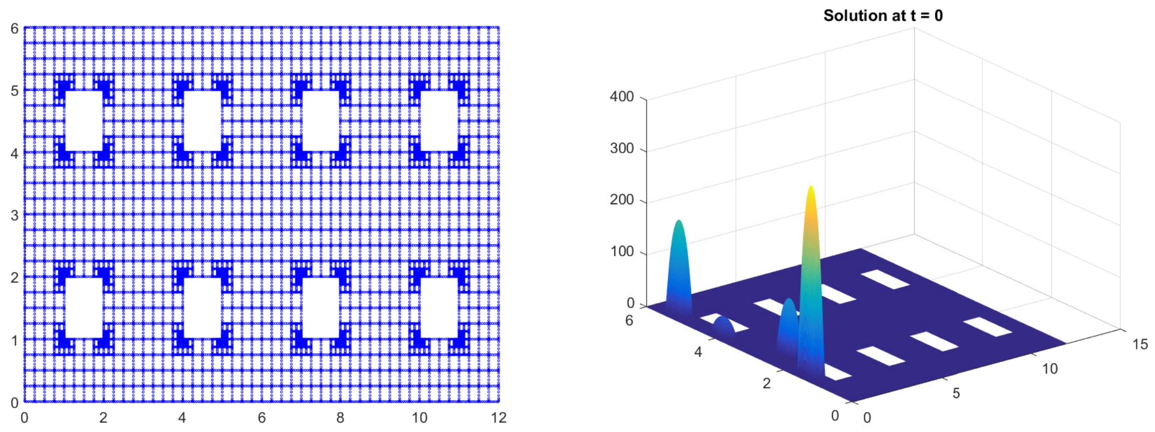


Figure 4.15: To the left: a coarsened version of the grid used for discretization. To the right: the initial data.

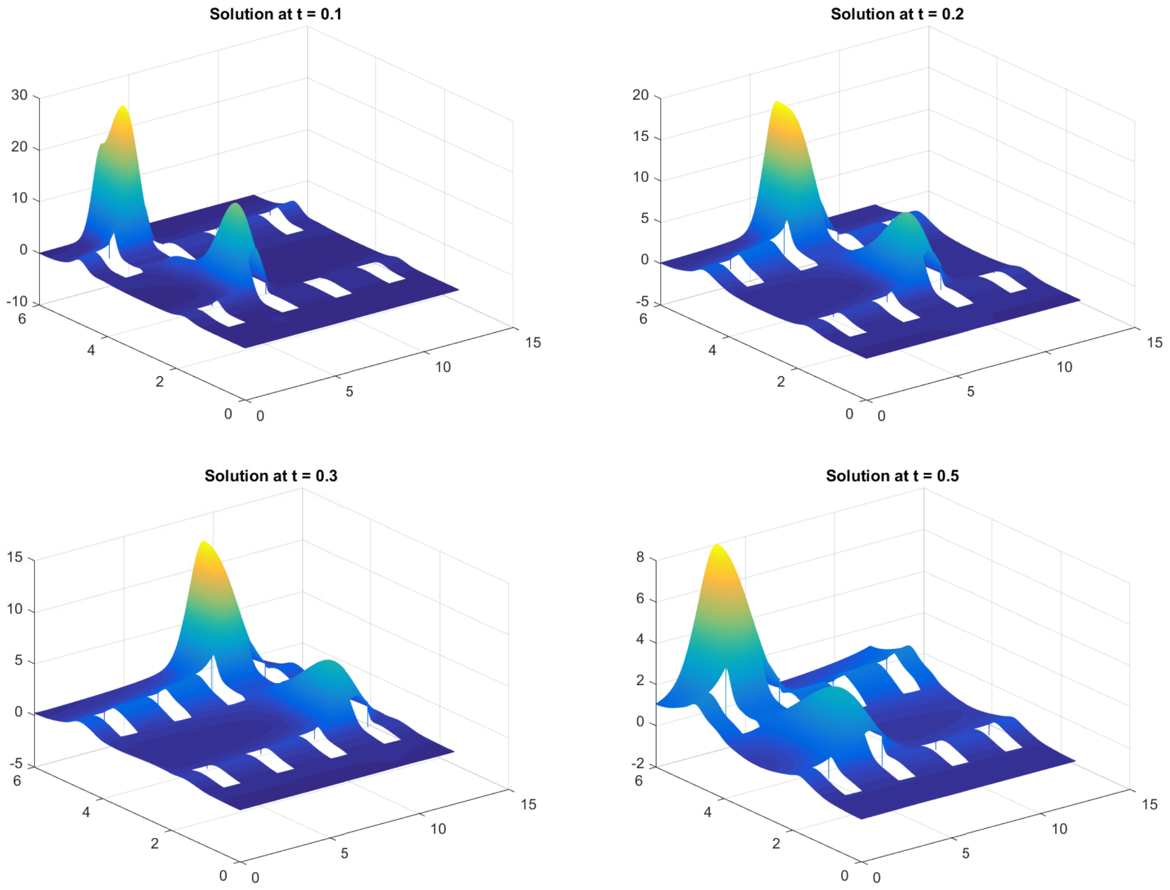


Figure 4.16: A sequence of snapshots of the solution for the advection diffusion problem. The final solution has traversed many times through the periodic domain.

Bibliography

[1] AmirHossein Aminfar, Sivaram Ambikasaran, and Eric Darve. A fast block low-rank dense solver with applications to finite-element matrices. Journal of Computational Physics, 304:170 – 188, 2016.

[2] T. Babb, A. Gillman, S. Hao, and P.G. Martinsson. An accelerated Poisson solver based on multidomain spectral discretization, 2016. Accepted for publication by BIT Numerical Mathematics; available as preprint arXiv:1612.02736.

[3] Mario Bebendorf. Hierarchical matrices, volume 63 of Lecture Notes in Computational Science and Engineering. Springer-Verlag, Berlin, 2008. A means to efficiently solve elliptic boundary value problems.

[4] Steffen Börm. Efficient numerical methods for non-local operators, volume 14 of EMS Tracts in Mathematics. European Mathematical Society (EMS), Zürich, 2010. $\mathcal{H}^2$-matrix compression, algorithms and analysis.

[5] Claudio Canuto, M Yousuff Hussaini, Alfio Quarteroni, and Thomas A Zang. Spectral methods: evolution to complex geometries and applications to fluid dynamics. Springer Science & Business Media, 2007.

[6] M. 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 Journal on Scientific Computing, 16(6):1241–1252, 1995.

[7] M. Causley, A. Christlieb, B. Ong, and L. Van Groningen. Method of lines transpose: An implicit solution to the wave equation. Mathematics of Computation, 83:2763–2786, 2014.

[8] T. Davis. Direct methods for sparse linear systems, volume 2. Siam, 2006.

[9] I.S. Duff, A.M. Erisman, and J.K. Reid. Direct Methods for Sparse Matrices. Oxford, 1989.

[10] B. Engquist and L. Ying. Sweeping preconditioner for the Helmholtz equation: hierarchical matrix representation. Communications on pure and applied mathematics, 64(5):697–735, 2011.

[11] B. Engquist and L. Ying. Sweeping preconditioner for the Helmholtz equation: moving perfectly matched layers. Multiscale Modeling & Simulation, 9(2):686–710, 2011.

[12] O.G. Ernst and M.J. Gander. Why it is difficult to solve Helmholtz problems with classical iterative methods. In Numerical analysis of multiscale problems, pages 325–363. Springer, 2012.

[13] Adrianna Geldermans, Peter; Gillman. An adaptive high order direct solution technique for elliptic boundary value problems, 2017. arXiv preprint #1710.08787.

[14] A. George. Nested dissection of a regular finite element mesh. SIAM J. on Numerical Analysis, 10:345–363, 1973.

[15] A. Gillman, A. Barnett, and P.G. Martinsson. A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media. BIT Numerical Mathematics, 55(1):141–170, 2015.

[16] A. Gillman and P. Martinsson. A direct solver with $O(N)$ complexity for variable coefficient elliptic pdes discretized via a high-order composite spectral collocation method. SIAM Journal on Scientific Computing, 36(4):A2023–A2046, 2014. arXiv.org report #1307.2665.

[17] Adrianna Gillman, Patrick Young, and Per-Gunnar Martinsson. A direct solver $o(n)$ complexity for integral equations on one-dimensional domains. Frontiers of Mathematics in China, 7:217–247, 2012. 10.1007/s11464-012-0188-3.

[18] Lars Grasedyck, Ronald Kriemann, and Sabine Le Borne. Domain decomposition based h-lu preconditioning. Numerische Mathematik, 112(4):565–600, 2009.

[19] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. J. Comput. Phys., 73(2):325–348, 1987.

[20] Leslie Greengard and Vladimir Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. In Acta numerica, 1997, volume 6 of Acta Numer., pages 229–269. Cambridge Univ. Press, Cambridge, 1997.

[21] W. Hackbusch, B. Khoromskij, and S. Sauter. On $\mathcal{H}^2$-matrices. In Lectures on Applied Mathematics, pages 9–29. Springer Berlin, 2002.

[22] Wolfgang Hackbusch. A sparse matrix arithmetic based on H-matrices; Part I: Introduction to H-matrices. Computing, 62:89–108, 1999.

[23] S Hao and PG Martinsson. A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré-Steklov operators. Journal of Computational and Applied Mathematics, 308:419 – 434, 2016.

[24] T. S. Haut, T. Babb, P. G. Martinsson, and B. A. Wingate. A high-order time-parallel scheme for solving wave propagation problems via the direct construction of an approximate time-evolution operator. IMA Journal of Numerical Analysis, 36(2):688–716, 2016.

[25] C. Kennedy and M. Carpenter. Additive Runge-Kutta schemes for convection-diffusion-reaction equations. Applied Numerical Mathematics, 44(1):139 – 181, 2003.

[26] B. Khoromskij and G. Wittum. Numerical solution of elliptic differential equations by reduction to the interface, volume 36. Springer, 2004.

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

[28] P.G. Martinsson. A composite spectral scheme for variable coefficient Helmholtz problems, 2012. arXiv preprint arXiv:1206.4136.

[29] P.G. Martinsson. A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method. Journal of Computational Physics, 242(0):460 – 479, 2013.

[30] P.G. Martinsson. The hierarchical poincare-steklov (hps) solver for elliptic pdes: A tutorial, 2015. arXiv preprint arXiv:1506.01308.

[31] P.G. Martinsson and V. Rokhlin. An accelerated kernel independent fast multipole method in one dimension. SIAM Journal of Scientific Computing, 29(3):1160–11178, 2007.

[32] Russell J. Hewett Natalie Beams, Adrianna Gillman. A parallel implementation of a high-order accurate solution technique for variable coefficient helmholtz problems, 2018. arXiv:1812.07167.

[33] H.P. Pfeiffer, L.E. Kidder, M.A. Scheel, and S.A. Teukolsky. A multidomain spectral method for solving elliptic equations. Computer physics communications, 152(3):253–273, 2003.

[34] R. Rosales, B. Seibold, D. Shirokoff, and D. Zhou. Order reduction in high-order Runge-Kutta methods for initial boundary value problems. ArXiv e-prints, December 2017.

[35] Yousef Saad. Iterative methods for sparse linear systems, volume 82. siam, 2003.

[36] Andrea Toselli and Olof Widlund. Domain decomposition methods-algorithms and theory, volume 34. Springer Science & Business Media, 2006.

[37] L.N. Trefethen. Spectral Methods in Matlab. SIAM, Philadelphia, 2000.

[38] U. Trottenberg, C.W. Oosterlee, and A. Schuller. Multigrid. Elsevier Science, 2000.

[39] J. Xia, S. Chandrasekaran, M. Gu, and X.S. Li. Fast algorithms for hierarchically semiseparable matrices. Numerical Linear Algebra with Applications, 17(6):953–976, 2010.

[40] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S. Li. Superfast multifrontal method for large structured linear systems of equations. SIAM J. Matrix Anal. Appl., 31(3):1382–1411, 2009.

[41] B. Yang and J.S. Hesthaven. Multidomain pseudospectral computation of Maxwell's equations in 3-d general curvilinear coordinates. Applied Numerical Mathematics, 33(1 – 4):281 – 289, 2000.

[42] L. Zepeda-Núñez, A. Scheuer, R. J. Hewett, and L. Demanet. The method of polarized traces for the 3D Helmholtz equation. ArXiv e-prints, January 2018.

[43] Luca Bergamaschi and Marco Vianello. Efficient computation of the exponential operator for large, sparse, symmetric matrices. Numer. Linear Algebra Appl., 7(1):27–45, 2000.

[44] G. Beylkin and K. Sandberg. Wave propagation using bases for bandlimited functions. Wave Motion, 41(3):263–291, 2005.

[45] W. J. Cody, G. Meinardus, and R. S. Varga. Chebyshev rational approximations to e^{-x} in $[0, +\infty)$ and applications to heat-conduction problems. J. Approximation Theory, 2:50–65, 1969.

[46] S.M. Cox and P.C. Matthews. Exponential time differencing for stiff systems. Journal of Computational Physics, 176(2):430 – 455, 2002.

[47] Anil Damle, Gregory Beylkin, Terry Haut, and Lucas Monzon. Near optimal rational approximations of large data sets. Applied and Computational Harmonic Analysis, 35(2):251 – 263, 2013.

[48] Laurent Demanet and Lexing Ying. Wave atoms and time upscaling of wave equations. Numer. Math., 113(1):1–71, 2009.

[49] I.S. Duff, A.M. Erisman, and J.K. Reid. Direct Methods for Sparse Matrices. Clarendon Press, Oxford, 1986.

[50] I. P. Gavrilyuk, W. Hackbusch, and B. N. Khoromskij. Hierarchical tensor-product approximation to the inverse and related operators for high-dimensional elliptic problems. Computing, 74(2):131–157, 2005.

[51] Stefan Guttel. Rational krylov approximation of matrix functions: Numerical methods and optimal pole selection. GAMM-Mitteilungen, 36(1):8–31, 2013.

[52] T. Haut and G. Beylkin. Fast and accurate con-eigenvalue algorithm for optimal rational approximations. SIAM Journal on Matrix Analysis and Applications, 33(4):1101–1125, 2012.

[53] T. S. Haut and B. A. Wingate. An asymptotic parallel-in-time method for highly oscillatory PDEs. SIAM J. of Sci. Comput., to appear. See also arXiv:1012.3196 [math.NA], 2013.

[54] N. Higham. The scaling and squaring method for the matrix exponential revisited. SIAM Journal on Matrix Analysis and Applications, 26(4):1179–1193, 2005.

[55] M. Hochbruck and C. Lubich. On Krylov subspace approximations to the matrix exponential operator. SIAM J. Numer. Anal., 34(5):1911–1925, 1997.

[56] Andrew J. Majda. Introduction to PDEs and waves for the atmosphere and ocean. Courant lecture notes in mathematics. Courant Institute of Mathematical Sciences Providence (R.I.), New York, 2003.

[57] P.G. Martinsson. A direct solver for variable coefficient elliptic {PDEs} discretized via a composite spectral collocation method. Journal of Computational Physics, 242(0):460 – 479, 2013.

[58] P.G. Martinsson. A direct solver for variable coefficient elliptic pdes discretized via a high-order composite spectral collocation method, a tutorial, 2013. arXiv.org report.

[59] Vladimir Maz'ya and Gunther Schmidt. On approximate approximations using gaussian kernels. IMA Journal of Numerical Analysis, 16:13–29, 1996.

[60] Vladimir Maz'ya and Gunther Schmidt. Approximate approximations, volume 141 of Mathematical Surveys and Monographs. American Mathematical Society, Providence, RI, 2007.

[61] Frank Müller and Werner Varnhorn. Error estimates for approximate approximations with Gaussian kernels on compact intervals. J. Approx. Theory, 145(2):171–181, 2007.

[62] Nathan Paldor and Andrey Sigalov. An invariant theory of the linearized shallow water equations with rotation and its application to a sphere and a plane. Dynamics of Atmospheres and Oceans, 51(1-2):26 – 44, 2011.

[63] Thomas Schmelzer and Lloyd N. Trefethen. Evaluating matrix functions for exponential integrators via Carathéodory-Fejér approximation and contour integrals. Electron. Trans. Numer. Anal., 29:1–18, 2007/08.