Spectral-in-time methods for time-evolution partial differential equations

Yiting Zhang

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2025

Reading Committee:

Tom Trogdon, Chair

Bernard Deconinc

Matthew Lorig

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

Abstract

Spectral-in-time methods for
time-evolution partial differential equations

Yiting Zhang

Chair of the Supervisory Committee:
Tom Trogdon
Applied Mathematics

In this thesis, we develop and analyze spectral-in-time methods to solve time-evolution partial differential equations (PDEs). To improve the computational complexity of solving the resulting linear system, we develop an algorithm using QR factorization in conjunction with the Woodbury matrix identity. Taking advantage of the intrinsic structure of the spectral discretizations of PDE problems, this algorithm can reduce the computational complexity compared to conventional direct solvers. However, as the dimension of the problem increases, solving the problem by applying the spectral-in-time method globally in time has a long execution time and requires more storage availability. To deal with these challenges, we propose an approach that embeds the spectral-in-time method within a time-stepping framework, which is similar to the classical time-stepping methods. This approach preserves high precision while considerably improving the computational efficiency by decomposing the entire temporal domain into manageable subintervals.

Moreover, we extend our method beyond linear PDEs by using Newton's method within the spectral-in-time framework for nonlinear PDEs. Instead of computing and storing the exact Jacobian matrices, we develop a final-time Jacobian matrix approach that generates much sparser Jacobian matrices. Through comprehensive numerical experiments, we show that this approach produces results indistinguishable in accuracy from the use of the exact Jacobian-based method with a slight reduction in the convergence of Newton's method.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank my advisor Tom Trogdon, whose guidance and unwavering support have been instrumental throughout my PhD journey. His mentorship has profoundly shaped my research direction, strengthened my presentations, and taught me what it truly means to be a dedicated researcher. It has been both an honor and a pleasure to learn from you.

I would also like to thank my committee members Bernard Deconinck, Matt Lorig, and Stefan Steinerberger for carefully reviewing my thesis. Their insightful questions and thoughtful discussions during my general and final exams greatly enriched my understanding and perspective. In addition, I offer my sincere thanks to Anne Greenbaum for her guidance during our reading sessions and her support during my exams.

I also express my gratitude to the members of the Applied Analysis Reading Group (AARG) and the Numerical Analysis Research Club (NARC) for their comments on my presentations and exciting weekly meetings, which have greatly expanded my research frontiers. I am also deeply grateful to my friends, fellow students, and postdocs in this department, who have made our academic community incredibly welcoming and pleasant.

And last but not least, I owe everything to my family who made me who I am today with all their love and support. And to my parents, thank you for your absolute faith in me, providing support through my academic and private ventures. To my wife, I cannot express enough gratitude for your endless support and patience, especially during my busiest moments. Lastly, I would like to thank my puppy Monet for always providing comfort and companionship when I needed it most.

Dedication

To my family, friends, and all the faculty members, whose unwavering support made this possible.

Chapter 1

Introduction

1.1 Motivation

Coefficient spectral methods [Boy01, OT13, TO15] provide an efficient framework for solving ordinary differential equations (ODEs) and partial differential equations (PDEs) with high precision. When applied to problems with sufficient smoothness, they achieve spectral or exponential convergence with relatively few degrees of freedom. A key advantage is their ability to represent differential operators as sparse or structured matrices, especially when appropriate basis functions are used.

However, these spectral discretizations often yield linear systems that are not strictly banded but rather almost banded. That is, the matrix can be written as $A = \tilde{A} + R$, where $\tilde{A}$ is a banded matrix and R is a low-rank matrix with nonzero entries confined to the first few rows (see Definition 2.1.1). Such systems arise naturally when boundary conditions are enforced in coefficient space or when dealing with variable coefficients and nonlinearities.

Standard direct solvers, such as LU or QR decomposition, do not take advantage of this special structure and lead to $\mathcal{O}(n^3)$ complexity, where n is the matrix size. Even classical banded solvers lose their efficiency [GL13], as the matrix R destroys the sparsity pattern. Therefore, efficient algorithms that can take advantage of both the banded structure of $\tilde{A}$ and the localized nature of R are needed.

Moreover, when solving time-evolution PDEs, traditional time-stepping methods can be inefficient for high-accuracy long-time integration, which is also the case for the conventional coefficient spectral method when applied globally in time. A fix of this is to divide the temporal domain into smaller subintervals, resulting in subproblems that can be dealt with using the coefficient spectral method efficiently. An application of this idea is to push the accuracy of the Tracy-Widom distribution [TW93, TW96] computation to machine precision (Section 5.5).

In nonlinear settings, the problem becomes even more delicate. Newton's method requires solving a sequence of linearized problems involving Jacobian matrices that can be extremely large and dense. For our spectral time-stepping approach, directly forming the Jacobian matrix leads to substantial fill-in, which leads to a dense matrix. One possible fix is to approximate the Jacobian matrix by evaluating it only at the final time, which

results in a sparse approximation that maintains the accuracy while dramatically reducing the storage requirement (Section 6.1).

These challenges motivate the development of fast, memory-efficient solvers for almost banded linear systems and their integration with traditional spectral methods for solving both linear and nonlinear time-evolution PDEs. For all the plots in this thesis, they are obtained by running on a computer with processor: 11th Gen Intel(R) Core(TM) i7-11800H 2.30GHz with 16.0GB of RAM.

1.2 Main contributions

First, we introduce a specialized QR algorithm that incorporates the Woodbury matrix identity to efficiently solve almost banded linear systems. This approach complements existing algorithms such as the right- or left-preconditioned restarted GMRES algorithms and the compressed QR algorithm in [OT13]. The resulting Woodbury QR algorithm (Algorithm 6) is designed to exploit the special structure of almost banded matrices and achieves a computational complexity comparable to existing ones (see Table 2.1). Throughout this thesis, the Woodbury QR algorithm is used as the default solver due to its relative simplicity in implementation and competitive performance.

Second, we propose a spectral time-stepping strategy for solving time-evolution PDEs, which embeds the spectral-in-time method within a time-stepping framework. Rather than solving a discretized linear system globally, which can become computationally infeasible when a large number of basis functions are required in the spatial domain, we partition the temporal domain into smaller subintervals and apply the spectral-in-time method locally to each subproblem. This approach maintains the high accuracy of the spectral-in-time method while reducing the computational cost. An application of the spectral time-stepping approach enables us to compute the well-known Tracy-Widom distribution [TW93, TW96] for arbitrary $\beta > 0$ to machine precision (Section 5.5).

Third, in addition to the Chebyshev basis used for finite nonperiodic spatial domains [TO15], we introduce two alternative spatial bases: the Fourier basis for finite periodic domains (Section 4.3.2) and a rational function basis for unbounded domains (Section 4.3.3). For each basis, we derive the corresponding differentiation and multiplication operators, showing that the resulting discretized linear systems preserve the almost banded structure or even become purely banded for finite periodic domains.

Fourth, we propose a more direct way to enforce the boundary conditions for time-evolution PDEs (Section 4.3), in contrast to the method used in [TO15]. In their approach, boundary conditions are enforced to the generalized Sylvester matrix equation to reduce the degrees of freedom and ensure that the resulting linear system,

expressed via Kronecker products, is well-posed. For our way of imposing the boundary conditions, inspired by the treatment of boundary conditions in the ODE case (Section 3.3), we enforce them directly to the top rows of the discretized linear system after Kronecker products are applied.

Lastly, we introduce a final-time Jacobian matrix approach to solve nonlinear time-evolution PDEs (Section 6.1). In each of Newton's iterations, instead of using the full (and typically dense) Jacobian matrix, we approximate it by evaluating the current iterate at the final time while keeping the right-hand side iterate unchanged. Although this approximation leads to only linear decay of the residuals, it yields a sparser Jacobian matrix and achieves an accuracy comparable to that obtained with the full Jacobian matrix. Notably, when the Fourier basis is used, the resulting Jacobian matrix using the final-time Jacobian matrix approach exhibits a banded structure.

1.3 Outline of the thesis

In Chapter 2, we introduce almost banded linear systems and present efficient algorithms to solve them. It begins with a motivation for their importance in spectral discretizations. After a formal definition, three algorithms are discussed: a left- or right-preconditioned restarted GMRES algorithm, a Woodbury QR algorithm, and a compressed QR algorithm. Their computational complexities are analyzed and compared. We then compare the Woodbury QR algorithm with the built-in solvers of `Julia` in terms of execution time and precision through some test problems. In the end, we conclude that the Woodbury QR algorithm is used. The `Julia` code of the Woodbury QR algorithm is provided.

In Chapter 3, we review the numerical methods that are used to solve linear ODEs. We then go over the logic behind the proof of the convergence of the computed Chebyshev coefficients to the truncated exact coefficients for solving linear ODEs using the ultraspherical spectral method [OT13]. The chapter concludes with numerical examples showing the performance of the ultraspherical spectral method on both linear and nonlinear problems along with the `Julia` code used.

In Chapter 4, we extend the spectral-in-time methods to solve linear time-evolution PDEs. In addition to the Chebyshev basis [TO15], which is used for finite nonperiodic spatial domains, we introduce two alternative spatial bases: the Fourier basis for finite periodic domains and a rational function basis for unbounded domains. The resulting discretized linear systems are almost banded and can be solved using the Woodbury QR algorithm. To further improve the computational efficiency, the spectral-in-time method is put in a time-stepping framework. Numerical experiments show that this spectral time-stepping approach maintains comparable accuracy while

reducing the execution time.

In Chapter 5, we study the Bloemendal–Virág equation [Blo11], whose solution is connected to the celebrated Tracy–Widom distribution for arbitrary $\beta > 0$. We begin by solving it using two traditional methods and then solve it using the spectral time-stepping approach introduced in Chapter 4.

In Chapter 6, we extend the spectral-in-time methods to deal with nonlinear time-evolution PDEs in conjunction with Newton's method. A final-time Jacobian matrix approach is proposed to reduce memory requirements and maintain matrix sparsity. Both the correct full Jacobian matrix and the final-time Jacobian matrix approaches are tested on six benchmark problems.

In Appendix A, we provide a brief tutorial of the `Julia` commands used in this thesis, mainly from `OperatorApproximation.jl` [TL24] and `ApproxFun.jl` [OGS+15].

Chapter 2

Almost banded matrices and relevant numerical solution strategies

The reason for introducing almost banded matrices and associated numerical methods in a complete chapter is the simple fact that they often arise when solving ODEs and PDEs. In particular, spectral methods usually result in linear systems with such a structure. Hence, it is important to look for and construct efficient algorithms capable of solving these linear systems. Prior to discussing about almost banded matrices, let us first discuss the simple concept of banded matrices.

Definition 2.0.1 (**Banded matrix**). *We call a matrix $A \in \mathbb{C}^{n\times n}$ a banded matrix if there are non-negative integers b_ℓ (lower bandwidth) and b_u (upper bandwidth) such that $a_{ij} = 0$ for $i - j > b_\ell$ and $j - i > b_u$. Then, the total bandwidth is $m = b_\ell + b_u + 1$.*

Direct methods such as pivoted LU and QR decompositions are used to solve the linear system $A\mathbf{x} = \mathbf{b}$ for invertible $A \in \mathbb{C}^{n\times n}$. However, these two methods generally require approximately $\mathcal{O}(n^3)$ floating-point operations (FLOPs). Although this cost may be acceptable for linear systems of small dimensions, it quickly becomes nasty for larger-scale problems. However, if A is banded as in Definition 2.0.1, specialized methods can significantly reduce the computational complexity for solving it. For example, applying Golub and Van Loan's so-called band-LU factorization [GL13, Section 4.3] can reduce the computational complexity to $\mathcal{O}(nb_\ell b_u)$. A similar reduction can also be obtained when using a similar version of QR factorization.

2.1 Almost banded matrices and related numerical methods

Now we are ready to give a precise definition of an almost banded matrix.

Definition 2.1.1 (**Almost banded matrix**). *A matrix $A \in \mathbb{C}^{n\times n}$ is almost banded if $A = \tilde{A} + R$, where $\tilde{A}$ is a (b_ℓ, b_u)-banded matrix. The rank of R is k, and its nonzero entries only appear in the first k rows with $k \ll n$ and $k \leq b_\ell + 1$. We call A a (b_ℓ, b_u, k)-almost banded matrix.*

When the matrix A is not truly banded but is almost banded, the LU and QR factorizations simply become inefficient, since R may introduce a large amount of fill-ins. Hence, this motivates the development of methods that are suitable for this class of matrices.

In the following sections, we discuss both direct and iterative methods that can solve almost banded linear systems efficiently. Specifically, we will examine the following three methods:

1. Restarted GMRES method with left/right preconditioning,
2. The Woodbury QR algorithm, and
3. The compressed QR algorithm.

2.2 Left/Right-preconditioned restarted GMRES algorithm

The Generalized Minimal Residual (GMRES) algorithm of Saad and Schultz [SS86] is a widely used Krylov subspace method for solving nonsymmetric linear systems of the form $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{C}^{n\times n}$ is invertible and $\mathbf{b} \in \mathbb{C}^n$. GMRES seeks an approximate solution $\mathbf{x}_n$ that minimizes the residual norm $\|\mathbf{b} - A\mathbf{x}_n\|_2$ over the Krylov subspace

$$\mathcal{K}_n(A, \mathbf{b}) = \text{span}(\mathbf{b}, A\mathbf{b}, \ldots, A^{n-1}\mathbf{b}). \tag{2.1}$$

This minimization is carried out using the Arnoldi algorithm [Arn51], which constructs an orthonormal basis for $\mathcal{K}_n(A, \mathbf{b})$.

2.2.1 Arnoldi algorithm and GMRES

To describe the GMRES algorithm, we first recall the Arnoldi algorithm (Algorithm 1) [Gre97], which generates an orthonormal basis $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$ for the Krylov subspace, which form the columns of Q_n. In addition, the Arnoldi algorithm also produces an upper Hessenberg matrix $H_{n+1,n} \in \mathbb{C}^{(n+1)\times n}$ such that $AQ_n = Q_{n+1}H_{n+1,n}$, where $Q_{n+1} = [Q_n, \mathbf{q}_{n+1}]$ and $H_{n+1,n} = [\mathbf{h}_1, \cdots, \mathbf{h}_n]$ for $\mathbf{h}_j = [h_{1,j}, \cdots, h_{j+1,j}, 0, \cdots, 0]^\intercal$, $j = 1, \cdots, n$. Using this relationship, GMRES solves a least-squares problem to find the vector $\mathbf{y} \in \mathbb{C}^n$ that minimizes $\|H_{n+1,n}\mathbf{y} - \|\mathbf{b}\|\mathbf{e}_1\|_2$, where $\mathbf{e}_1$ is the first standard basis vector in $\mathbb{C}^{n+1}$. Once $\mathbf{y}$ is found, the approximate solution is updated according to $\mathbf{x}_n = Q_n\mathbf{y}$.

A standard approach for computing the vector $\mathbf{y}$ is by applying Givens rotations [Giv58] to the matrix $H_{n+1,n}$, converting it into the product of a unitary matrix $F^* \in \mathbb{C}^{(n+1)\times(n+1)}$ and an upper triangular matrix $T \in$

$\mathbb{C}^{(n+1)\times n}$, where the final row of T only consists of zeros. In this way, $\mathbf{y}$ can be found by solving the upper triangular system $T_{n\times n}\mathbf{y} = \|\mathbf{b}\| (F\mathbf{e}_1)_{n\times 1}$, where $T_{n\times n}$ is the top $n \times n$ block of T and $(F\mathbf{e}_1)_{n\times 1}$ is the top n entries of the first column of F. Therefore, the 2-norm of the residual is just the bottom entry of $\|\mathbf{b}\|F\mathbf{e}_1$.

Algorithm 1 Arnoldi algorithm

Input: Matrix $A \in \mathbb{C}^{n\times n}$, initial vector $\mathbf{q}_1$ with $\|\mathbf{q}_1\| = 1$.
for $j = 1, 2, 3, \ldots, n$ **do**
  $\tilde{\mathbf{q}}_{j+1} = A\mathbf{q}_j$
  for $i = 1, \ldots, j$ **do**
    $h_{i,j} = \langle \tilde{\mathbf{q}}_{j+1}, \mathbf{q}_i \rangle$
    $\tilde{\mathbf{q}}_{j+1} \leftarrow \tilde{\mathbf{q}}_{j+1} - h_{i,j}\mathbf{q}_i$
  end for
  $h_{j+1,j} = \|\tilde{\mathbf{q}}_{j+1}\|$
  $\mathbf{q}_{j+1} = \tilde{\mathbf{q}}_{j+1}/h_{j+1,j}$
end for

The high-level version of the GMRES algorithm is given in Algorithm 2. For a more detailed variant, see [Gre97, Algorithm 3].

Algorithm 2 GMRES algorithm

Input: Matrix $A \in \mathbb{C}^{n\times n}$, right-hand side vector $\mathbf{b}$.
$\mathbf{q}_1 = \mathbf{b}/\|\mathbf{b}\|$.
for $n = 1, 2, 3, \ldots$ **do**
  Perform step n of Algorithm 1 on A and $\mathbf{q}_n$ to obtain $H_{n+1,n}$, Q_n, and $\mathbf{q}_{n+1}$.
  $\mathbf{y} = \arg\min_{\mathbf{y}'} \|H_{n+1,n}\mathbf{y}' - \|\mathbf{b}\|\mathbf{e}_1\|_2$.
  $\mathbf{x}_n = Q_n\mathbf{y}$.
end for

2.2.2 Restarted GMRES

For large-scale problems, the dimension of the Krylov subspace can increase substantially, leading to high memory and computational costs. A common fix is restarted GMRES. Instead of letting the Krylov subspace expand indefinitely, restarted GMRES (denoted GMRES(t) for a restart parameter t) truncates the subspace after t iterations. Specifically, the approximate solution at restart becomes the initial guess for the next cycle of GMRES. Although this approach reduces memory demands, it can cause stagnation if t is not large enough.

2.2.3 Preconditioning

An important way to speed up Krylov subspace methods is through preconditioning. Given a non-singular matrix M such that $M^{-1}A$ or AM^{-1} has a condition number smaller than A, we can rewrite $A\mathbf{x} = \mathbf{b}$ in one of the

following equivalent ways:

1. Left preconditioning: $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$.

2. Right preconditioning: $(AM^{-1})(M\mathbf{x}) = \mathbf{b}$.

In both ways, the operator being applied in GMRES, $M^{-1}A$ or AM^{-1}, has a more clustered spectrum than A itself, allowing faster convergence. Computing M exactly as A^{-1} is usually infeasible in situations where iterative methods are considered. An effective approach is the incomplete LU (ILU) factorization [Saa03], which is similar to a complete LU factorization except certain entries in L and U are dropped to reduce the level of fill-in. The degree of incompleteness can be controlled by a threshold parameter τ, balancing computational cost against factorization accuracy.

For an almost banded matrix A, ILU factorization can remain expensive due to the existence of the matrix R. To avoid this issue, let us write $A = \tilde{A} + R$ as in Definition 2.1.1, where $\tilde{A}$ is invertible. Instead of applying ILU on A, one can apply it on $\tilde{A}$ and use the resulting ILU factorization as a preconditioner. Provided that the ILU factorization of $\tilde{A}$ is numerically robust, this approach typically produces an effective and stable preconditioner, significantly improving computational efficiency.

In the following, we show the high-level outlines of the restarted GMRES with both left-preconditioned and right-preconditioned. In the left-preconditioned case, the initial residual is $\mathbf{r}_0 = M^{-1}(\mathbf{b} - A\mathbf{x}_0)$ [Saa03], while in the right-preconditioned variant, the residual computation remains $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, but the iterative updates pass through M^{-1}.

Algorithm 3 Left-preconditioned restarted GMRES(t)

Input: Almost banded matrix A, right-hand side vector $\mathbf{b}$, restart parameter t, initial guess $\mathbf{x}_0$.
Extract the banded part $\tilde{A}$ of A and compute ILU factorization $\tilde{A} \approx LU$ with tolerence τ.
$M = LU$.
for $i = 1, 2, 3, \ldots$ until convergence **do**
  $\mathbf{r}_0 = M^{-1}(\mathbf{b} - A\mathbf{x}_0)$.
  $\mathbf{q}_1 = \mathbf{r}_0/\|\mathbf{r}_0\|$.
  for $j = 1, \ldots, t$ **do**
    Perform step j of Algorithm 1 to $(M^{-1}A, \mathbf{q}_j)$ to obtain $H_{j+1,j}$, Q_j, and $\mathbf{q}_{j+1}$.
    Obtain $\tilde{H}_{j+1,j}$ through applying Givens rotations to $H_{j+1,j}$.
    Solve $\tilde{H}_{j+1,j}\mathbf{y}_j = \|\mathbf{r}_0\|\mathbf{e}_1$.
    $\mathbf{x}_j = \mathbf{x}_0 + Q_j\mathbf{y}_j$.
    if the residual is sufficiently small **then**
      break
    end if
  end for
  $\mathbf{x}_0 = \mathbf{x}_j$.
end for

Algorithm 4 Right-preconditioned restarted GMRES(t)

Input: Almost banded matrix A, right-hand side vector $\mathbf{b}$, restart parameter t, initial guess $\mathbf{x}_0$.
Extract the banded part $\tilde{A}$ of A and compute ILU factorization $\tilde{A} \approx LU$ with tolerence τ.
$M = LU$.
for $i = 1, 2, 3, \ldots$ until convergence **do**
  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$.
  $\mathbf{q}_1 = \mathbf{r}_0 / \|\mathbf{r}_0\|$.
  for $j = 1, \ldots, t$ **do**
    Perform step j of Algorithm 1 to $(AM^{-1}, \mathbf{q}_j)$ to obtain $H_{j+1,j}$, Q_j, and $\mathbf{q}_{j+1}$.
    Obtain $\tilde{H}_{j+1,j}$ through applying Givens rotations to $H_{j+1,j}$.
    Solve $\tilde{H}_{j+1,j}\mathbf{y}_j = \|\mathbf{r}_0\|\mathbf{e}_1$.
    $\mathbf{x}_j = \mathbf{x}_0 + M^{-1}Q_j\mathbf{y}_j$.
    if the residual is sufficiently small **then**
      break
    end if
  end for
  $\mathbf{x}_0 = \mathbf{x}_j$.
end for

The computational complexity of Algorithm 3 and 4 can be summarized in the following lemma.

Lemma 2.2.1. *Let $A = \tilde{A} + R \in \mathbb{C}^{n\times n}$, where $\tilde{A}$ is a (b_ℓ, b_u)-banded invertible matrix with total bandwidth $m = b_\ell + b_u + 1$, and R is a rank-k matrix with nonzero entries confined to the first k rows. The computational complexity to solve the linear system $A\mathbf{x} = \mathbf{b}$ using restarted GMRES(t), preconditioned with Crout's ILU factorization of $\tilde{A}$, is given by $\mathcal{O}\left(nm^2 + Nnt(m+t)\right)$, where N denotes the total number of outer iterations.*

Proof. For Algorithm 3, the preconditioner is constructed using Crout's ILU factorization on $\tilde{A}$, which requires $\mathcal{O}(nm^2)$ FLOPs. This cost comes from processing $\mathcal{O}(n)$ columns, each requiring $\mathcal{O}(m^2)$ operations due to the banded structure, resulting in lower and upper triangular matrices L and U that each maintains $\mathcal{O}(m)$ bandwidth.

At each outer iteration, the computation of $\mathbf{b} - A\mathbf{x}_0$ requires $\mathcal{O}((m+k)n) \approx \mathcal{O}(mn)$ FLOPs. This includes $\mathcal{O}(mn)$ operations from the matrix-vector multiplication with the banded matrix $\tilde{A}$, and $\mathcal{O}(kn)$ from the rank-k correction. Applying the preconditioner by solving $LU^\intercal\mathbf{z}_0 = \mathbf{r}_0$ requires two banded triangular solves, each with complexity $\mathcal{O}(mn)$.

In each iteration, the dominant cost comes from applying the Arnoldi algorithm and Givens rotations. Each application of the preconditioned operator $M^{-1}A\mathbf{q}_j$ consists of three steps. First, the matrix-vector product $A\mathbf{q}_j$ introduces a cost of $\mathcal{O}((m+k)n) \approx \mathcal{O}(mn)$. Second, solving for $M^{-1}A\mathbf{q}_j$ requires $\mathcal{O}((b_\ell+b_u)n) \approx \mathcal{O}(mn)$ operations as both L and U maintain the $\mathcal{O}(m)$ bandwidth like $\tilde{A}$. Third, the orthogonalization

against t existing Krylov basis vectors introduces an additional $\mathcal{O}(tn)$ cost. Furthermore, the processing of the Hessenberg matrix via Givens rotations contributes $\mathcal{O}(t^2)$ FLOPs per restart cycle.

Multiplying the per-iteration cost $\mathcal{O}((m+t)n)$ by the total number of iterations Nt, and adding the one-time preconditioner setup cost $\mathcal{O}(nm^2)$, yields the overall computational complexity. The complexity analysis of Algorithm 4 follows analogously. □

2.3 The Woodbury QR algorithm

Similar to the previous section, in which the preconditioner is constructed only from the banded part of the matrix A, the Woodbury QR method is based on the decomposition $A = \tilde{A} + R$. This decomposition allows the matrix R to be expressed as $R = UV$, where $U \in \mathbb{C}^{n\times k}$ contains columns that correspond to the first k standard basis vectors in $\mathbb{C}^n$ and $V \in \mathbb{C}^{k\times n}$.

For an invertible matrix A, the Woodbury matrix identity [Woo50] provides an efficient approach for computing the inverse of A:

$$A^{-1} = (\tilde{A} + UV)^{-1} = \tilde{A}^{-1} - \tilde{A}^{-1}U\left(I_k + V\tilde{A}^{-1}U\right)^{-1}V\tilde{A}^{-1}, \tag{2.2}$$

where I_k denotes the $k \times k$ identity matrix. From this identity, one obtains the solution to $A\mathbf{x} = \mathbf{b}$ as

$$\mathbf{x} = A^{-1}\mathbf{b} = \tilde{A}^{-1}\mathbf{b} - \tilde{A}^{-1}U(I_k + V\tilde{A}^{-1}U)^{-1}V\tilde{A}^{-1}\mathbf{b}. \tag{2.3}$$

Denote $\mathbf{x}_b = \tilde{A}^{-1}\mathbf{b}$ and $\mathbf{x}_U = \tilde{A}^{-1}U$. Then (2.3) can be written more compactly as $\mathbf{x} = \mathbf{x}_b - \mathbf{x}_U(I_k + V\mathbf{x}_U)^{-1}V\mathbf{x}_b$.

To efficiently compute $\mathbf{x}_b$ and $\mathbf{x}_U$, one can take advantage of the banded structure of $\tilde{A}$ by using a sparse QR factorization. Algorithm 5 provides a high-level description of this process, where Givens rotations are applied to introduce zeros. With the banded matrix $\tilde{A}$ and Algorithm 5, we can then derive the Woodbury QR algorithm as in Algorithm 6.

This approach takes advantage of the banded structure of $\tilde{A}$ to perform QR factorization efficiently and then uses the Woodbury matrix identity to handle the rank-k perturbation. It is often highly efficient when $k \ll n$, as the computational complexity associated with the dense matrix $(I_k + V\tilde{A}^{-1}U)$ remains relatively small.

The computational complexity of Algorithm 6 can be summarized in the following lemma.

Lemma 2.3.1. *Let $A = \tilde{A} + UV \in \mathbb{C}^{n\times n}$ be a (b_ℓ, b_u, k)-almost banded matrix, where A is invertible. The computational complexity to solve $A\mathbf{x} = \mathbf{b}$ using the Woodbury QR algorithm is $\mathcal{O}(nb_\ell m + nb_u k + nk^2) \approx \mathcal{O}(nm^2)$.*

Algorithm 5 QR algorithm for banded linear systems

Input: (b_ℓ, b_u)-banded matrix $A \in \mathbb{C}^{n\times n}$, right-hand side(s) $\mathbf{b} \in \mathbb{C}^{n\times p}$.
for $j = 1$ **to** n **do**
  for $i = j+1$ **to** $\min(b_\ell + j, n)$ **do**
    if entry (i, j) of A is nonzero **then**
      Apply a Givens rotation to zero out the (i, j) entry of A and $\mathbf{b}$.
    end if
  end for
end for
Solve the resulting upper-triangular system in place.
return $\mathbf{b}$

Algorithm 6 Woodbury QR algorithm

Input: (b_ℓ, b_u, k)-almost banded matrix $A \in \mathbb{C}^{n\times n}$, right-hand side(s) $\mathbf{b} \in \mathbb{C}^{n\times p}$.
Construct $U \in \mathbb{C}^{n\times k}, V \in \mathbb{C}^{k\times n}$ such that $A = \tilde{A} + UV$.
$U_b = [\,U, \mathbf{b}\,]$.
Apply Algorithm 5 on $\left(\tilde{A}, U_b, b_\ell\right)$ to update U_b in place.
$\mathbf{x}_U = U_b[:, 1\!:\!k]$ and $\mathbf{x}_b = U_b[:, k+1]$.
Compute $C = I_k + V\mathbf{x}_U$.
$\mathbf{x}_s = V\,\mathbf{x}_b$.
$\mathbf{x}_s \leftarrow C^{-1}\mathbf{x}_s$.
$\mathbf{x}_b \leftarrow \mathbf{x}_b - \mathbf{x}_U\,\mathbf{x}_s$.
$\mathbf{b} \leftarrow \mathbf{x}_b$.
return $\mathbf{b}$

Proof. The computation consists of three parts. First, the QR factorization of $\tilde{A}$ requires $\mathcal{O}(nb_\ell m)$ FLOPs: each of the n columns requires eliminating b_ℓ subdiagonal entries through Givens rotations, with each rotation modifying m elements within the band. Simultaneously updating the augmented matrix $U_b = [U, \mathbf{b}]$ costs an additional $\mathcal{O}(k^2 b_\ell)$ FLOPs.

Second, the back substitution step solves the upper-triangular system with $k+1$ right-hand sides. Each row requires $\mathcal{O}(b_u)$ FLOPs per right-hand side vector, totaling $\mathcal{O}(nb_u k)$ FLOPs.

Finally, the Woodbury computations involve: 1) the matrix product VX_U at $\mathcal{O}(nk^2)$ FLOPs, 2) solving a $k \times k$ system via LU factorization ($\mathcal{O}(k^3)$), and 3) vector updates costing $\mathcal{O}(nk)$ FLOPs.

Combining these components yields $\mathcal{O}(nb_\ell m + nb_u k + nk^2) \approx \mathcal{O}(nm^2)$ as $k \le m$ and $b_\ell, b_u \sim \mathcal{O}(m)$. □

2.4 The compressed QR algorithm

Another non-iterative method for efficiently solving almost banded linear systems was proposed by Olver and Townsend in [OT13, Section 5]. Its complexity is $\mathcal{O}(nm^2)$. In contrast to a naive QR factorization via Givens

rotations, which typically leads to significant fill-in, their approach exploits the special structure of almost banded matrices to significantly reduce the computational complexity.

Consider the decomposition $A = \tilde{A} + R$ as in Definition 2.1.1, and let $\mathcal{R}$ denote the submatrix consisting of the first k dense rows of R. After applying a QR factorization via Givens rotations to A, we obtain an upper-triangular matrix B and an updated right-hand side vector $\mathbf{c}$. The critical observation by Olver and Townsend is that the fill-in portion of B, specifically the entries B_{ij}, for $1 \leq i \leq n-m$, $n-m+i-1 \leq j \leq n$, can each be represented as linear combinations of the rows of $\mathcal{R}$. Consequently, each filled-in row of B can be compactly represented using at most m original banded terms plus an additional k-dimensional coefficient vector derived from the rows of $\mathcal{R}$.

Due to this compressed representation, the back substitution stage is slightly modified. While the last m rows of the upper-triangular system can be solved by conventional means, each of the first $(n-m)$ rows involves an additional term related to the dense rows in $\mathcal{R}$. In particular, each equation for row i, with $1 \leq i \leq n-m$, takes the form:

$$B_{i,i}x_i = c_i - \sum_{j=1}^{m-1} B_{i,i+j}x_{i+j} - \mathbf{b}_k^\top \sum_{j=i+m}^{n} \mathcal{R}_{:,j}x_j, \tag{2.4}$$

where $\mathbf{b}_k$ is the corresponding k-dimensional coefficient vector encoding the dependency of the ith row on $\mathcal{R}$.

A high-level description of this approach is summarized in Algorithm 7, hereafter referred to as the compressed QR algorithm. The computational complexity of Algorithm 7 can be summarized in the following lemma, which is in agreement with the conclusion in [OT13].

Lemma 2.4.1. *Let $A = \tilde{A} + R \in \mathbb{C}^{n\times n}$ be a (b_ℓ, b_u, k)-almost banded matrix. The computational complexity to solve $A\mathbf{x} = \mathbf{b}$ using the compressed QR algorithm is $\mathcal{O}(nb_\ell m + nb_\ell k) \approx \mathcal{O}(nm^2)$.*

Proof. The QR factorization phase involves $\mathcal{O}(nb_\ell)$ Givens rotations, each acting on two rows of the matrix. For each rotation, applying the rotation to the banded portion, which affects m entries in each of the two rows, costs $\mathcal{O}(m)$ FLOPs. Updates to the right-hand side vector $\mathbf{b}$ cost $\mathcal{O}(1)$ FLOPs, while updates to the dense rows and coefficient vectors $\mathbf{b}_k^{(i)}$ and $\mathbf{b}_k^{(j)}$ (each of length k) contribute an additional $\mathcal{O}(k)$ FLOPs. Thus, the total cost of the QR factorization phase is $\mathcal{O}(nb_\ell m + nb_\ell k)$.

Following factorization, the back substitution step begins with solving the last m rows of the upper triangular system, which costs $\mathcal{O}(m^2)$ FLOPs. For each of the remaining $(n-m)$ rows, computing the compressed solution involves three components: a matrix-vector product involving a k-dimensional coefficient vector ($\mathcal{O}(k)$ FLOPs), a banded inner product of length $(m-1)$ ($\mathcal{O}(m)$ FLOPs), and a final scalar update

Algorithm 7 Compressed QR algorithm

Input: (b_ℓ, b_u, k)-almost banded matrix $A \in \mathbb{C}^{n\times n}$, $\mathcal{R} \in \mathbb{C}^{k\times n}$, $\mathbf{b} \in \mathbb{C}^n$.
Initialize $\mathbf{b}_k^{(i)} = \mathbf{0}$ for all $i = 1, \ldots, n$.
for $j = 1$ **to** n **do**
  for $i = j + 1$ **to** $\min(j + b_\ell,\, n)$ **do**
    if $A_{i,j} \neq 0$ **then**
      Form Givens rotation $G_{(i,j)}$ to zero out $A_{i,j}$.
      $(A_{j,*}, A_{i,*}) \leftarrow G_{(i,j)} \cdot (A_{j,*}, A_{i,*})$ and $(b_j, b_i) \leftarrow G_{(i,j)} \cdot (b_j, b_i)$.
      Update $\mathbf{b}_k^{(j)}, \mathbf{b}_k^{(i)}$.
    end if
  end for
end for
Denote the resulting upper-triangular matrix by B and the updated right-hand side by $\mathbf{c}$.
$\mathbf{p}_{n-m+1} \leftarrow \mathbf{0}$
for $i = n - m$ **to** 1 **do**
  $\mathbf{p}_i \leftarrow \mathbf{u}_{i+m}\, \mathcal{R}_{:,\, i+m} \;+\; \mathbf{p}_{i+1}$
  $x_i \leftarrow \dfrac{1}{B_{i,i}} \left(\mathbf{c}_i \;-\; \displaystyle\sum_{j=1}^{m-1} B_{i,\, i+j}\, x_{\,i+j} \;-\; \mathbf{b}_k^{(i)\top}\, \mathbf{p}_i \right)$
end for
return $\mathbf{x}$

($\mathcal{O}(k)$ FLOPs). This yields a total of $\mathcal{O}(nm + nk)$ FLOPs.

Combining both phases, the total complexity is $\mathcal{O}(nb_\ell m + nb_\ell k) \approx \mathcal{O}(nm^2)$ for $b_\ell \sim \mathcal{O}(m)$ and $k \leq m$. □

Remark 1. *Another approach for solving almost banded linear systems was introduced by Chandrasekaran and Gu [CG03]. Their method employs two-sided Givens rotations applied to the almost banded matrix A, effectively avoiding fill-in caused by the dense row(s). With the same inputs of Algorithm 7, this method also achieves a computational complexity of $\mathcal{O}(nm^2)$.*

2.5 Optimal matrix size

When applying spectral-in-time methods to solve ODEs and PDEs, one usually ends up solving almost banded linear systems whose dimension n directly affects both the accuracy and the computational complexity. Increasing n usually leads to improved numerical accuracy, but at the same time introduces an increase in memory usage and computational cost. Hence, for any specified level of precision, it is important to determine an optimal problem size n_{opt} that can achieve the required precision while maintaining competitive computational complexity.

As an example, we can identify n_{opt} by applying the Woodbury QR algorithm adaptively. Suppose n_{opt} is approximately known. Then, starting with a small dimensional problem, one can successively double the matrix

dimension (e.g., from 2 to 4, 8, etc.) until around n_{opt}. Considering a worst-case scenario, where the final doubling step results in a dimension just below n_{opt} (e.g., $n_{\text{opt}} - 1$), thus requiring an additional doubling step, the total computational complexity of determining n_{opt} via adaptively applying the Woodbury QR algorithm is estimated as

$$2m^2 n_{\text{opt}} + m^2 n_{\text{opt}} + m^2 \frac{n_{\text{opt}}}{2} + \cdots + 2m^2 = 2m^2 n_{\text{opt}} \sum_{k=0}^{\log_2(n_{\text{opt}})} \frac{1}{2^k} \leq 2m^2 n_{\text{opt}} \sum_{k=0}^{\infty} \frac{1}{2^k} = 4m^2 n_{\text{opt}}. \tag{2.5}$$

Thus, performing an adaptive search to identify the optimal dimension n_{opt} requires a computational complexity of the same order as a single execution, which differs only by a constant factor.

2.6 Comparison with the built-in solvers of `Julia`

Table 2.1 summarizes the computational complexities of the three approaches we have discussed in solving alomst banded linear system $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{C}^{n\times n}$ is defined as in Definition 2.0.1 and 2.1.1. Now, let us do some

Method	**Computational Complexity**	**Use Case**
Pre-rGMRES(t)	$\mathcal{O}(nm^2 + Nnt(m+t))$	An efficient preconditioner can be computed.
Woodbury QR	$\mathcal{O}(nm^2)$	All cases
Compressed QR	$\mathcal{O}(nm^2)$	All cases

Table 2.1: Comparison of different solvers for $A\mathbf{x} = \mathbf{b}$, where $A = \tilde{A} + R$

numerical experiments to evaluate the performance of the Woodbury QR algorithm. A very important point is that to accurately compare numerical methods in terms of either accuracy or execution time, we need to write our own algorithms instead of using any built-in solvers. However, one may wonder how the Woodbury QR algorithm performs compared to the built-in solvers. As mentioned in the introduction, since all of our numerical experiments are done using `Julia`, we can benchmark the Woodbury QR algorithm against the built-in LU, QR, and backslash solvers.

We will compare these methods in two settings: one with double-precision arithmetic (Float64) and the other with 76-digit-precision arithmetic (BigFloat). To assess their performance, we construct the test linear system by generating the $n \times n$ almost banded matrix A, the right-hand side vector $\mathbf{b}$, and the solution vector $\mathbf{x}$ according to Algorithm 8, whose outputs are all in sparse format, which means that if an entry is zero, then it will not be stored. In contrast, a matrix in dense format means that all entries are stored even for zero entries. Note that the random almost banded matrix A is constructed with entries following the Chi distribution, which is used to

minimize the possibility of getting singular matrices.

Algorithm 8 Construction of A, $\mathbf{b}$, and $\mathbf{x}$

1: **Input:** Integers n, df, b_ℓ, b_u, k.
2: Initialize a random sparse $n \times n$ banded matrix A with entries following the $\chi(df)$ distribution with lower and upper bandwidths b_ℓ and b_u
3: Add a random sparse $k \times n$ random sparse matrix to the first k rows of A, whose entries also follow the $\chi(df)$ distribution
4: Generate a random sparse $n \times 1$ solution vector $\mathbf{x}$ with entries following the $\chi(df)$ distribution
5: Compute $\mathbf{b} \leftarrow A\mathbf{x}$
6: **Output:** A, $\mathbf{b}$, $\mathbf{x}$

The `Julia` code for Algorithm 8 is as follows.

```
function generate_data(n, df, bl, bu, k)
    chi_sampler = Chi(df)
    A = sparse(BandedMatrix(rand(chi_sampler, n, n), (bl, bu)))
    A[1:k, :] .+= sparse(rand(chi_sampler, k, n))
    x = sparse(rand(chi_sampler, n, 1)); b = A * x
    return A, b, x
end
```

To run the Woodbury QR algorithm, we generate $\tilde{A}$, the banded part of A, the augmented matrix Ub, and the matrix V using Algorithm 9.

Algorithm 9 Construction of $\tilde{A}$, **Ub**, and **V**

1: **Input:** A, n, $\mathbf{b}$, k, b_u.
2: Initialize V as a $k \times n$ zero matrix
3: **for** $i = 1$ **to** k **do**
4: **for** $j = b_u + i + 1$ **to** n **do**
5: temp$= A_{i,j}$
6: $A_{i,j} = 0$
7: $V_{i,j} =$ temp
8: **end for**
9: **end for**
10: Form the $n \times (k+1)$ augmented matrix **Ub**
11: **Output:** A, **Ub**, **V**

The `Julia` code for Algorithm 9 is as follows.

```
function generate_data2(A, n, b, k, bu)
    V = spzeros(k,n)
    for i = 1:k
        for j = bu + i + 1:n
            temp = A[i, j]; A[i,j] = 0; V[i,j] = temp
        end
    end
    Ub = Matrix([sparse(1:k, 1:k, ones(k), n, k) b])
    return dropzeros!(A), Ub, V
end
```

For the following numerical experiments, we set the number of dense rows k to $\lceil n/300 \rceil$. For b_ℓ and b_u, we set them at $n/100$. For the degrees of freedom df of the Chi distribution, we set it to 10 for all cases. The execution time is measured using the `BenchmarkTools` package [Rev15], which is used to accurately measure the average time it takes to complete a task. The precision of an algorithm is measured by $\|\mathbf{x} - \tilde{\mathbf{x}}\|$, where $\mathbf{x}$ is the exact result and $\tilde{\mathbf{x}}$ is the computed result.

Method / n	Built-in LU	Built-in QR	Backslash	Woodbury QR
10^3 (Float64)	71.645	104.030	0.022	0.523
2×10^3 (Float64)	1161.286	1495.945	0.101	8.865
3×10^3 (Float64)	4814.142	6808.311	0.210	31.387
10^3 (BigFloat)	158.114	343.990	83.327	4.774
2×10^3 (BigFloat)	1330.827	2675.958	622.229	78.668
3×10^3 (BigFloat)	19925.811	17683.344	31578.221	510.480

Table 2.2: Comparison of the execution times for different solvers in `Julia`. All times are measured in seconds.

We can see that all algorithms achieve comparable accuracy. In terms of execution time, the built-in backslash solver outperforms the rest in double-precision arithmetic and the Woodbury QR algorithm is the second best. This result is expected since the built-in solvers did not take advantage of the special structure of the almost banded matrix.

However, the computational efficiency of backslash is lost in BigFloat arithmetic, as it cannot be applied

Method / n	Built-in LU	Built-in QR	Backslash	Woodbury QR
10^3 (Float64)	$5.313(-11)$	$2.354(-11)$	$3.498(-11)$	$4.805(-9)$
2×10^3 (Float64)	$3.735(-10)$	$1.482(-10)$	$1.602(-10)$	$3.784(-9)$
3×10^3 (Float64)	$4.918(-10)$	$2.128(-10)$	$2.911(-10)$	$1.129(-8)$
10^3 (BigFloat)	$7.984(-72)$	$3.800(-72)$	$7.984(-72)$	$3.795(-71)$
2×10^3 (BigFloat)	$2.499(-71)$	$4.864(-71)$	$2.499(-71)$	$1.291(-70)$
3×10^3 (BigFloat)	$1.310(-70)$	$6.504(-71)$	$1.310(-70)$	$4.901(-70)$

Table 2.3: Comparison of the precisions for different solvers in `Julia`.

directly on a matrix in its sparse format. The matrix must first be converted to its dense format before backslash is applied. We can see that the Woodbury QR algorithm shows a superior execution speed in this scenario.

In conclusion, if one is trying to solve a linear system $A\mathbf{x} = \mathbf{b}$, where the matrix A is stored in its sparse format and double-precision arithmetic provides sufficient accuracy, the built-in backslash solver is generally optimal. However, caution must be taken, as the standard pivoting step is skipped to preserve the sparsity pattern of A for indefinite matrix A, and the computed result will be affected. Of course, if the matrix is well-conditioned or a preconditioner is relatively easy to obtain, one can try a left- or right-preconditioned restarted GMRES algorithm. Otherwise, for a matrix with a special pattern, one can try to write a tailored optimized algorithm to take advantage of this special structure, and it usually outperforms the built-in solvers. From now on, we will use the Woodbury QR algorithm since it is more straightforward to implement than the compressed QR algorithm and has comparable computational complexity.

Before presenting the `Julia` code of the Woodbury QR algorithm, we need the following function to assess whether a certain entry in a matrix in its sparse format is zero without assigning that entry to 0.

```
function nz_ind(A::SparseMatrixCSC{Tv, Ti}, row::Int, col::Int) where {Tv, Ti<:Integer}
    for idx in A.colptr[col]:(A.colptr[col + 1] - 1)
        if  A.rowval[idx] == row
            return true
        end
    end
    return false
end
```

The Julia code for the Woodbury QR algorithm is as follows.

```
function sparse_qr(A, b, bl)
    m = size(A, 1); n = size(b, 2)
    for j = 1:m
        for i = j+1:min(bl+j, m)
            if nz_ind(A, i, j)
                r = hypot(A[j, j], A[i, j]); c = abs(A[j, j]) / r; s = c*A[i, j] / A[j, j]
                for k = j:m
                    if nz_ind(A, j, k) || nz_ind(A, i, k)
                        tmpp = c * A[j, k] + conj(s) * A[i, k]
                        A[i, k] = -s * A[j, k] + c * A[i, k]; A[j, k] = tmpp
                    end
                end
                SparseArrays.dropzeros!(A)
                for k = 1:n
                    tmpp = c * b[j, k] + conj(s) * b[i, k]
                    b[i, k] = -s * b[j, k] + c * b[i, k]; b[j, k] = tmpp
                end
            end
            SparseArrays.dropstored!(A, i, j)
        end
    end
    b .= A \ b
end
```

```
function woodbury_sparseqr(B, Ub, V, k::Int, bl)
    sparse_qr(B, Ub, bl); C = I + V*view(Ub,:,1:k)
    xs = V*view(Ub,:,k + 1:k + 1); xs .= C\xs
    view(Ub,:,k + 1:k + 1) .-= view(Ub,:,1:k)*xs
    return view(Ub,:,k + 1:k + 1)
end
```

Chapter 3

Review of numerical methods for solving linear and nonlinear ODEs

Before moving on to the idea of the spectral-in-time methods for solving time-evolution PDEs, it is helpful to briefly revisit some numerical approaches commonly used for solving ODEs. Among the classical ones, finite difference (FD) methods remain the most popular. FD methods approximate derivatives using finite differences computed at discrete grid points, a concept with historical origins dating back to the foundational calculus developed by Newton and Leibniz. Due to their conceptual simplicity and straightforward implementation, FD methods are widely used in practice and often serve as benchmarks against which other numerical methods are evaluated.

Let us assume we are trying to solve the following initial-value problem (IVP):

$$\mathbf{u}'(t) = \mathbf{f}(\mathbf{u}(t), t), \quad t \in [t_0, t_M], \tag{3.1}$$

subject to the initial condition

$$\mathbf{u}(t_0) = \mathbf{c}, \tag{3.2}$$

where, in general, $\mathbf{u}(t)$ is an $(N+1)$-dimensional vector with components $u_0(t), \ldots, u_N(t)$, and $\mathbf{c} \in \mathbb{C}^{N+1}$. The vector-valued function $\mathbf{f}(\mathbf{u}(t), t)$ has components $f_0(\mathbf{u}(t), t), \ldots, f_N(\mathbf{u}(t), t)$, each potentially nonlinear and dependent on all components of $\mathbf{u}(t)$.

(3.1) is linear if $\mathbf{f}(\mathbf{u}(t), t) = A(t)\mathbf{u} + \mathbf{g}(t)$, where $A(t) \in \mathbb{C}^{(N+1)\times(N+1)}$ and $\mathbf{g}(t) \in \mathbb{C}^{N+1}$. Numerical methods for solving linear IVPs rely mainly on various forms of time-stepping methods. Although these methods can broadly be viewed as finite-difference approximations, they are grouped into two principal classes: multistage methods and linear multistep methods. Given the importance of these two classes of methods for solving PDEs, we now present an overview of them.

3.1 Multistage methods

The family of Runge–Kutta (RK) methods [Run95, Kut01], serving as a canonical example of multistage methods, computes intermediate approximations of both the solution and its derivative at several stages within each time step. Let h denote the time step and let $U^n \approx \mathbf{u}(t_n)$ at t_n. Then, a s-stage RK method can be expressed as:

$$\mathbf{k}_i = f\left(t_n + c_i h,\ U^n + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j\right), \quad i = 1, \ldots, s, \tag{3.3}$$

$$U^{n+1} = U^n + h\sum_{i=1}^{s} b_i \mathbf{k}_i, \tag{3.4}$$

where the coefficients a_{ij}, b_i, and c_i are arranged in the well-known Butcher tableau:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} \tag{3.5}$$

Depending on the structure of (3.5), RK methods can be classified as explicit RK (ERK) methods, implicit RK (IRK) methods, and diagonally implicit RK (DIRK) methods.

For ERK methods, the elements on and above the diagonal in the a_{ij} portion of (3.5) are all equal to zero, that is, $a_{ij} = 0$ for $j \geq i$. With an explicit method, each of the $\mathbf{k}_i$ values is computed using only the previously computed $\mathbf{k}_j$ values. ERK methods are particularly advantageous for solving non-stiff systems (see, for example, [LeV07, Section 8.2] for the definition of stiffness), especially when computational efficiency is a primary concern. However, ERK methods do not exhibit A-stability [Dah63], thus restricting their effectiveness for stiff equations. The following Butcher tableaux show three examples of ERK methods.

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array}$$

Midpoint method (order 2)

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & 1/6 & 2/3 & 1/6 \end{array}$$

Kutta's method (order 3)

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array} \tag{3.6}$$

RK4 method (order 4)

For IRK methods, each stage vector $\mathbf{k}_i$ generally depends on all the stage vectors $\mathbf{k}_j$, including those with indices

$j \geq i$; this corresponds to $a_{ij} \neq 0$ for only some $j > i$. Consequently, IRK methods can be computationally expensive for solving systems of ODEs. Specifically, solving an ODE system of dimension r using an s-stage IRK method requires solving a system of sr equations to determine the s stage vectors $\mathbf{k}_i$. However, IRK methods are able to handle stiff problems due to their superior stability properties: the Gauss-Legendre family of IRK methods are A-stable. This makes them particularly suitable for stiff systems with rapidly decaying modes, such as chemical kinetics [LeV07, Section 7.4.1] or parabolic PDEs [LeV07, Chapter 9]. The following Butcher tableaux show two examples of IRK methods.

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$$

Gauss-Legendre method (order 2)

$$\begin{array}{c|cc} 1/2-\sqrt{3}/6 & 1/4 & 1/4-\sqrt{3}/6 \\ 1/2+\sqrt{3}/6 & 1/4+\sqrt{3}/6 & 1/4 \\ \hline & 1/2 & 1/2 \end{array} \tag{3.7}$$

Gauss-Legendre method (order 4)

One subclass of IRK methods that are simpler to implement are the DIRK methods, in which $\mathbf{k}_i$ depends on $\mathbf{k}_j$ for $j \leq i$, that is, $a_{ij} = 0$ for $j > i$ and $a_{ii} \neq 0$. DIRK methods can be designed to maintain A-stability for stiff systems, and certain implementations can even achieve the stronger L-stability [HW96]. The following Butcher tableaux show two examples of DIRK methods, and the Crouzeix's method shown below is a notable example that is L-stable.

$$\begin{array}{c|cc} 1/4 & 1/4 & 0 \\ 3/4 & 1/2 & 1/4 \\ \hline & 1/2 & 1/2 \end{array}$$

Qin and Zhang's method (order 2)

$$\begin{array}{c|cc} 1/2+\sqrt{3}/6 & 1/2+\sqrt{3}/6 & 0 \\ 1/2-\sqrt{3}/6 & -\sqrt{3}/3 & 1/2+\sqrt{3}/6 \\ \hline & 1/2 & 1/2 \end{array} \tag{3.8}$$

Crouzeix's method (order 3)

3.2 Linear multistep methods

Linear multistep methods (LMMs) [Dah56] take advantage of solution information from previous time steps to predict or correct the solution in the following time steps. Again, let h denote the time step and let $U^n \approx \mathbf{u}(t_n)$ at t_n. Then, a r-step LMM can be expressed as:

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = h \sum_{j=0}^{r} \beta_j f\left(U^{n+j}, t_{n+j}\right). \tag{3.9}$$

(3.9) is explicit if $\beta_r = 0$, otherwise implicit. Two important LMM families are Adams methods and backward differentiation formula (BDF) methods [CH52]. Adams methods construct solutions through polynomial inter-

polation of derivatives from previous time steps. The explicit Adams method, called Adams-Bashforth method [Bas83], has the form:

$$U^{n+r} = U^{n+r-1} + h\sum_{j=0}^{r-1} \beta_j f(U^{n+j}, t_{n+j}). \tag{3.10}$$

Since Adams–Bashforth methods are explicit, they are preferred for non-stiff problems. However, as with most explicit methods, Adams–Bashforth methods are not A-stable and thus suffer from stability restrictions for stiff systems. The following shows two examples of Adams-Bashforth methods:

$$\text{2-step:} \quad U^{n+2} = U^{n+1} + \frac{h}{2}\left(3f(U^{n+1}, t_{n+1}) - f(U^n, t_n)\right), \tag{3.11}$$

and

$$\text{4-step:} \quad U^{n+4} = U^{n+3} + \frac{h}{24}\left(55f(U^{n+3}, t_{n+3}) - 59f(U^{n+2}, t_{n+2}) + 37f(U^{n+1}, t_{n+1}) - 9f(U^n, t_n)\right). \tag{3.12}$$

The implicit Adams method, called Adams-Moulton method [Mou26], improves the order of accuracy by 1 by setting $\beta_r \neq 0$:

$$U^{n+r} = U^{n+r-1} + h\sum_{j=0}^{r} \beta_j f(U^{n+j}, t_{n+j}). \tag{3.13}$$

Being implicit, Adams–Moulton methods offer improved stability properties over their Adams–Bashforth counterparts, making them more suitable for moderately stiff problems. However, each step generally requires solving a (non)linear system, which can be computationally expensive for large-scale or strongly stiff systems. The following shows two examples of Adams-Moulton methods.

$$\text{1-step:} \quad U^{n+1} = U^n + \frac{h}{2}\left(f(U^n, t_n) + f(U^{n+1}, t_{n+1})\right), \tag{3.14}$$

which is the only A-stable Adams method due to Dahlquist's second barrier theorem [Dah63], and

$$\text{3-step:} \quad U^{n+3} = U^{n+2} + \frac{h}{24}\left(f(U^n, t_n) - 5f(U^{n+1}, t_{n+1}) + 19f(U^{n+2}, t_{n+2}) + 9f(U^{n+3}, t_{n+3})\right), \tag{3.15}$$

whih is used in moderately stiff systems.

Another important class of LMMs is the family of BDF methods, which are particularly well-suited for stiff

systems by approximating derivatives via backward differences:

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = h\beta_r f(U^{n+r}, t_{n+r}). \tag{3.16}$$

An r-step BDF method achieves rth-order accuracy, though zero-stability [Dah63] restricts practical use to $r \leq 6$. Lower-order forms include:

$$\begin{aligned} r = 1: &\quad U^{n+1} - U^n = hf(U^{n+1}, t_{n+1}) \quad \text{(Backward Euler)}, \\ r = 2: &\quad 3U^{n+2} - 4U^{n+1} + U^n = 2hf(U^{n+2}, t_{n+2}), \text{ and} \\ r = 3: &\quad 11U^{n+3} - 18U^{n+2} + 9U^{n+1} - 2U^n = 6hf(U^{n+3}, t_{n+3}). \end{aligned} \tag{3.17}$$

While not A-stable for $r > 2$, BDF methods exhibit A(α)-stability [Wid67] with angles α decreasing as r increases:

$$\begin{aligned} r = 1, 2: &\quad \alpha = 90^\circ \qquad r = 3: \quad \alpha = 88^\circ \\ r = 4: &\quad \alpha = 73^\circ \qquad r = 5, 6: \quad \alpha = 51^\circ,\ 18^\circ \end{aligned} \tag{3.18}$$

Being A-stable or A(α)-stable makes them ideal for stiff systems like semiconductor device modeling [BRF83]. The absolute stability region [Dah63] contracts with larger r, which requires careful time-step selection.

3.3 Coefficient spectral methods for solving linear ODEs

Spectral methods for solving differential equations can be broadly classified into two frameworks: the collocation methods [Ors72, Tre00, Boy01, DH16, Tro24] and the coefficient methods [Boy01, OT13].

The collocation methods operate at discrete nodal points (e.g., Chebyshev-Gauss-Lobatto nodes) by enforcing the differential equation exactly at these points. The derivatives are approximated using differentiation matrices $\mathbf{D}$ constructed using derivatives of the basis functions. For Lagrange interpolation polynomials $\{\ell_j(x)\}_{j=1}^N$ defined on collocation nodes $\{x_i\}_{i=0}^N$, the differentiation matrix $\mathbf{D}$ has entries

$$D_{ij} = \ell_j'(x_i), \quad \ell_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{N} \frac{x - x_k}{x_j - x_k}, \quad i, j = 0, 1, \ldots, N. \tag{3.19}$$

Here, i indexes the nodes at which derivatives are computed, and j indexes the basis functions. While Lagrange polynomials are standard, alternative bases are used in specialized cases: Fourier bases for periodic problems enable

pseudo-spectral Fourier differentiation; Hermite polynomials suit unbounded domains. Although collocation methods are straightforward to implement, the resulting linear systems become increasingly ill-conditioned as the degree of the polynomial increases.

On the other hand, the coefficient spectral (Galerkin) method can be classified into three main categories based on the choice of basis and test functions:

1. Ritz-Galerkin method solves differential equations by working with symmetric and energy-minimizing systems. Usually, Legendre or Chebyshev polynomials are used as bases. This approach uses identical basis functions for both trial and test spaces, creating stable solutions for systems where energy minimization principles apply, such as Poisson's equation [BS08, Section 0.2].

2. Bubnov-Galerkin method uses identical basis functions, usually the Chebyshev polynomials, for both trial and test spaces. This approach generates dense, upper-triangular systems, as demonstrated in the classical Tau-method for solving boundary value problems (BVPs) [Lan38, Boy01].

3. Petrov-Galerkin method uses different bases (e.g., Chebyshev for trial functions and ultraspherical polynomials for test functions) to achieve sparse, banded systems [She95, MS00, OT13].

In this chapter, we will focus on the ultraspherical spectral method [OT13], which belongs to the family of Petrov-Galerkin methods, by approximating the solution using the first n Chebyshev polynomials of the first kind $\{T_k(t)\}_{k=0}^{n-1}$:

$$u(t) \approx \sum_{k=0}^{n-1} u_k T_k(t). \tag{3.20}$$

Following the notations in [OT13], consider an Nth-order linear ODE with variable coefficients for $t \in [-1, 1]$:

$$\mathcal{L}u(t) = f(t), \quad \mathcal{B}\mathbf{u} = \mathbf{c}, \tag{3.21}$$

where $\mathcal{L}$ is an Nth-order linear ordinary differential operator (ODO) defined as

$$\mathcal{L}u(t) = \sum_{i=0}^{N} a^i(t) u^{(i)}(t), \tag{3.22}$$

with coefficient functions $a^0, \ldots, a^N$, and the right-hand side function f being sufficiently smooth over the domain $[-1, 1]$. The boundary operator $\mathcal{B}$ enforces N boundary conditions on the coefficient vector $\mathbf{u} = [u_0, u_1, \cdots, u_{n-1}]^\intercal$, $\mathbf{c} \in \mathbb{C}^N$. We assume throughout that the differential equation is not singular, meaning

that the leading-coefficient function $a^N(t)$ does not vanish on the interval $[-1, 1]$.

The ultraspherical spectral method constructs an almost banded system with three fundamental operators: the kth-order differentiation operator $\mathcal{D}_k$, the multiplication operator $\mathcal{M}_k[a^i]$, and the conversion operator $\mathcal{S}_k$.

The derivative of the Chebyshev polynomials satisfies

$$\frac{dT_n(t)}{dt} = \begin{cases} nC_{n-1}^{(1)}(t), & n \geq 1, \\ 0, & n = 0, \end{cases} \tag{3.23}$$

where $C_{n-1}^{(1)}(x)$ is the ultraspherical polynomial [PK90] with parameter 1 of degree $n-1$. The weight function of the ultraspherical polynomials $C_n^{(m)}(x)$ is given by $\left(1-x^2\right)^{m-\frac{1}{2}}$. More generally, due to the differentiation rule of ultraspherical polynomials:

$$\frac{dC_n^{(m)}(t)}{dt} = 2mC_{n-1}^{(m+1)}(t), \tag{3.24}$$

we have

$$\frac{d^kT_n(t)}{dt^k} = \begin{cases} 2^{k-1}n\,(k-1)!C_{n-k}^{(k)}(t), & n \geq k, \\ 0, & 0 \leq n \leq k-1. \end{cases} \tag{3.25}$$

In matrix form, the kth-order differentiation operator is given by

$$\mathcal{D}_k = 2^{k-1}(k-1)! \begin{pmatrix} \overbrace{0 \quad \cdots \quad 0}^{k} & k & & & \\ & & k+1 & & \\ & & & k+2 & \\ & & & & \ddots \end{pmatrix}, \quad k \geq 1. \tag{3.26}$$

For $k \geq 1$, $\mathcal{D}_k$ maps a vector of Chebyshev coefficients to a vector of $C^{(k)}$ coefficients of the kth derivative. $\mathcal{D}_0$ is the identity operator.

The multiplication operator $\mathcal{M}_0[a^i]$ for $k = 0$ encodes the variable coefficient function $a^i(t)$ as a Toeplitz-plus-Hankel matrix acting on the Chebyshev coefficients that takes the vector of Chebyshev coefficients of $u(t)$

and returns the vector of Chebyshev coefficients of $a^i(t)u(t)$ [BT97]. In matrix form, $\mathcal{M}_0[a^i]$ is given by

$$\mathcal{M}_0[a^i] = \frac{1}{2}\left[\begin{pmatrix} 2a_0 & a_1 & a_2 & a_3 & \cdots \\ a_1 & 2a_0 & a_1 & a_2 & \ddots \\ a_2 & a_1 & 2a_0 & a_1 & \ddots \\ a_3 & a_2 & a_1 & 2a_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ a_1 & a_2 & a_3 & a_4 & \cdots \\ a_2 & a_3 & a_4 & a_5 & \iddots \\ a_3 & a_4 & a_5 & a_6 & \iddots \\ \vdots & \iddots & \iddots & \iddots & \iddots \end{pmatrix}\right], \tag{3.27}$$

where

$$a^i(t) = \sum_{j=0}^{\infty} a_j T_j(t). \tag{3.28}$$

In practice, $\mathcal{M}_0[a^i]$ is banded with a bandwidth of m, which is determined by truncating the series in (3.28) using only the first m coefficients so that the Chebyshev coefficients a_j, for $j \geq m$, are or nearly of machine precision, which is doable since $a^i(t)$ is assumed to be continuous with bounded variation. More generally, $\mathcal{M}_k[a^i]$ represents the multiplication of a^i in the $C^{(k)}$ space. The explicit form of $\mathcal{M}_k[a^i]$ can be found in [OT13, Section 3.1]. In [Tow14, Chapter 6], it is shown that $\mathcal{M}_k[a^i]$ satisfies a three-term recurrence relation.

Remark 2. *For ultraspherical spectral methods, only functions with finite basis expansions (e.g., degree-d polynomials with $m \sim \mathcal{O}(d)$) have bandwidth-m multiplication matrices independent of n. General functions require m to grow with n. The key assumption in this chapter is that the coefficient functions are all polynomials of degree $d \ll n$ so that the bandwidth m is independent of n.*

Regarding the conversion operator $\mathcal{S}_k$, since the ultraspherical polynomials satisfy the recurrence relation for $m \geq 1$:

$$C_k^{(m)} = \begin{cases} \frac{m}{m+k}\left(C_k^{(m+1)} - C_{k-2}^{(m+1)}\right), & k \geq 2, \\ \frac{m}{m+1} C_1^{(m+1)}, & k = 1, \\ C_0^{(m+1)}, & k = 0, \end{cases} \tag{3.29}$$

and for the special case $m = 0$:

$$T_k = \begin{cases} \frac{1}{2}\left(C_k^{(1)} - C_{k-2}^{(1)}\right), & k \geq 2, \\ \frac{1}{2} C_1^{(1)}, & k = 1, \\ C_0^{(1)}, & k = 0, \end{cases} \tag{3.30}$$

we obtain the conversion operators $\mathcal{S}_0$ and $\mathcal{S}_k$ given explicitly by:

$$\mathcal{S}_0 = \begin{pmatrix} 1 & 0 & -\frac{1}{2} & & \\ & \frac{1}{2} & 0 & -\frac{1}{2} & \\ & & \frac{1}{2} & 0 & \ddots \\ & & & \frac{1}{2} & \ddots \\ & & & & \ddots \end{pmatrix}, \quad \mathcal{S}_k = \begin{pmatrix} 1 & 0 & -\frac{k}{k+2} & & \\ & \frac{k}{k+1} & 0 & -\frac{k}{k+3} & \\ & & \frac{k}{k+2} & 0 & \ddots \\ & & & \frac{k}{k+3} & \ddots \\ & & & & \ddots \end{pmatrix}. \tag{3.31}$$

Here, the operator $\mathcal{S}_0$ converts a vector of Chebyshev coefficients into a vector of $C^{(1)}$-coefficients, while $\mathcal{S}_k$ converts a vector of $C^{(k)}$-coefficients into a vector of $C^{(k+1)}$-coefficients. Note that for $k \geq 1$, the dense structure of $\mathcal{S}_0^{-1}\mathcal{S}_1^{-1}\cdots\mathcal{S}_{k-1}^{-1}\mathcal{D}_k$, which represents the kth-order differentiation matrix in the Chebyshev basis, indicates the advantage of using the ultraspherical spectral method over the Bubnov-Galerkin method using Chebyshev polynomials of the first kind as the only basis functions.

With the above operators, we can combine them to discretize (3.21) while preserving sparsity. Let the $n \times \infty$ operator $\mathcal{P}_n$ be as follows:

$$\mathcal{P}_n = (I_n, \mathbf{0}), \tag{3.32}$$

where I_n is the square identity matrix of size n. The linear ODO

$$\mathcal{L} = \mathcal{M}_N[a^N]\mathcal{D}_N + \sum_{i=0}^{N-1} \mathcal{S}_{N-1}\cdots\mathcal{S}_i\mathcal{M}_i[a^i]\mathcal{D}_i \tag{3.33}$$

and the boundary operator $\mathcal{B}$ form an almost banded linear system after truncation. In matrix form, we have

$$A_n \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} \mathbf{c} \\ \\ \\ \mathcal{P}_{n-K}\mathcal{S}_{N-1}\cdots\mathcal{S}_0\mathbf{f} \end{pmatrix}, \quad A_n = \begin{pmatrix} \mathcal{B}\mathcal{P}_n^\intercal \\ \\ \mathcal{P}_{n-N}\mathcal{L}\mathcal{P}_n^\intercal \end{pmatrix}, \tag{3.34}$$

where $\mathbf{f} = [f_0, f_1, \cdots]^\intercal$ is the vector containing the Chebyshev coefficients of the right-hand side function $f(t)$.

The convergence of the computed coefficient vector $\mathbf{u}_n$ to the first n terms of the true coefficient vector $\mathbf{u}$ is established through a sequence of lemmas that make use of the Banach space ℓ^2_λ defined below. In the following, we define key components and outline the logical flow of the proof.

First of all, the Banach space ℓ^2_λ is defined by the norm:

$$\|\mathbf{u}\|_{\ell^2_\lambda} = \sqrt{\sum_{k=0}^{\infty} |u_k|^2 (k+1)^{2\lambda}} < \infty. \tag{3.35}$$

Besides, a special preconditioner operator is defined as follows:

$$\mathcal{R} = \frac{1}{2^{N-1}(N-1)!} \mathsf{diag}\left(\overbrace{1 \quad \cdots \quad 1}^{N}, \frac{1}{N}, \frac{1}{N+1}, \cdots \right). \tag{3.36}$$

For D being the order of the ODE problem, Lemma 3.3.1 below shows that the preconditioned operator $\binom{\mathcal{B}}{\mathcal{L}}\mathcal{R}$ can be written as a compact perturbation of the identity operator.

Lemma 3.3.1 (Lemma 4.3 in [OT13]). *Suppose that the boundary operator $\mathcal{B} : \ell^2_D \to \mathbb{C}^N$ is bounded. Then $\binom{\mathcal{B}}{\mathcal{L}}\mathcal{R} = I + \mathcal{K}$, where $\mathcal{K} : \ell^2_\lambda \to \ell^2_\lambda$ is compact for $\lambda = D-1, D, \cdots$.*

The key takeaway of this lemma is that the operator $\mathcal{K}$ is compact, and thus $I + \mathcal{K}$ is a Fredholm operator, which allows us to show uniqueness and well-conditionedness by Lemma 3.3.2.

Lemma 3.3.2 (Lemma 4.4 in [OT13]). *Suppose that $\binom{\mathcal{B}}{\mathcal{L}} : \ell^2_{\lambda+1} \to \ell^2_\lambda$ is an invertible operator for some $\lambda \in \{D, D-1, \cdots\}$. Then, as $n \to \infty$, $\|A_n R_n\|_{\ell^2_\lambda} = \mathcal{O}(1)$ and $\|(A_n R_n)^{-1}\|_{\ell^2_\lambda} = \mathcal{O}(1)$, for the diagonal matrix $R_n = \mathcal{P}_n \mathcal{R} \mathcal{P}_n^\intercal$ and the truncated matrix*

$$A_n = \mathcal{P}_n \begin{pmatrix} \mathcal{B} \\ \mathcal{L} \end{pmatrix} \mathcal{P}_n^\intercal. \tag{3.37}$$

Finally, Theorem 3.3.1 shows that the $\ell^2_{\lambda+1}$-norm of the difference between the true coefficients $\mathbf{u}$ and the numerical coefficients $\mathbf{u}_n$ padded with zeros is bounded above by the truncation error of the true coefficients.

Theorem 3.3.1 (Theorem 4.5 in [OT13]). *Suppose $\mathbf{f} \in \ell^2_{\lambda-N+1}$ for some $\lambda \in \{D-1, D, \cdots\}$, and that $\binom{\mathcal{B}}{\mathcal{L}} : \ell^2_{\lambda+1} \to \ell^2_\lambda$ is an invertible operator. Define*

$$\mathbf{u}_n = A_n^{-1} \mathcal{P}_n \begin{pmatrix} \mathbf{c} \\ \\ \\ \mathcal{S}_{N-1} \cdots \mathcal{S}_0 \mathbf{f} \end{pmatrix}. \tag{3.38}$$

Then $\|\mathbf{u} - \mathcal{P}_n^{\mathsf{T}}\mathbf{u}_n\|_{\ell^2_{\lambda+1}} \leq C\|\mathbf{u} - \mathcal{P}_n^{\mathsf{T}}\mathcal{P}_n\mathbf{u}\|_{\ell^2_{\lambda+1}} \to 0.$

In cases where the true coefficient vector $\mathbf{u}$ is unknown, if the numerical coefficients $\mathbf{u}_n$ form a Cauchy sequence in $\ell^2_{\lambda+1}$ space, completeness guarantees convergence to a limit $\mathbf{u}^* \in \ell^2_{\lambda+1}$, and the uniqueness of the solution (from Lemma 3.3.2) forces $\mathbf{u}^* = \mathbf{u}$.

Upon solving (3.34), the solution is approximated as in (3.20). The discretized matrix A_n in (3.34) retains a banded structure with $\mathcal{O}(m)$ bandwidth, except for the first K rows. This is exactly what we refer to as an almost banded matrix by Definition 2.1.1, and thus (3.34) can be solved in $\mathcal{O}(nm^2)$ operations using the Woodbury QR algorithm discussed in Section 2.3.

3.4 Coefficient spectral methods for solving nonlinear ODEs

To apply the ultraspherical spectral method to an Nth-order nonlinear ODE for $t \in [-1, 1]$, we consider the problem

$$\mathcal{L}u(t) = f(t) + F\left(t, u, u_t, \cdots, u_{mt}\right), \quad \mathcal{B}\mathbf{u} = \mathbf{c}, \tag{3.39}$$

where F is a given nonlinear function depending on $u(t)$ and its spatial derivatives up to order m, $\mathcal{B}$ enforces N boundary conditions on the Chebyshev coefficient vector $\mathbf{u} = [u_0, u_1, \ldots, u_{n-1}]^{\mathsf{T}}$, and $\mathbf{c} \in \mathbb{C}^N$ represents the boundary data. The linear operator $\mathcal{L}u(t)$ maintains the structure previously defined in (3.21).

To address the nonlinear problem, we use Newton's method. Defining the nonlinear operator as

$$\mathcal{N}\left(t, u, u_t, \cdots, u_{mt}\right) = \mathcal{L}u(t) - f(t) - F\left(t, u, u_t, \cdots, u_{mt}\right), \tag{3.40}$$

Newton's iteration is based on solving

$$\mathcal{N}\left(t, u + \delta u, u_t + \delta u_t, \cdots, u_{mt} + \delta u_{mt}\right) \approx \mathcal{N}\left(t, u, u_t, \cdots, u_{mt}\right) + \mathcal{J}\left(t, u, u_t, \cdots, u_{mt}\right)\delta u = 0, \tag{3.41}$$

leading to the linearized equation

$$\mathcal{J}\left(t, u, u_t, \cdots, u_{mt}\right)\delta u = \mathcal{L}\delta u - \sum_{j=0}^{m} \frac{\partial F}{\partial u_{jt}}\delta u_{jt} = f(t) - \mathcal{L}u(t) + F\left(t, u, u_t, \cdots, u_{mt}\right), \tag{3.42}$$

where $\mathcal{J}\left(t, u, u_t, \cdots, u_{mt}\right)$ is the Jacobian operator. To initialize Newton's method, we first compute an initial

guess $u_0(t)$ by solving the linear problem

$$\mathcal{L}u_0(t) = f(t), \quad \mathcal{B}\mathbf{u}_0 = \mathbf{c}. \tag{3.43}$$

We then iterate for $k = 0, 1, 2, \ldots$, solving the linearized equation for the increment $\delta u^{(k)}$:

$$\left(\mathcal{L} - \sum_{j=0}^{m} \frac{\partial F}{\partial u_{jt}} \mathcal{D}_j\right) \delta u^{(k)} = f(t) - \mathcal{L}u_k(t) + F\left(t, u, u_t, \cdots, u_{mt}\right), \quad \mathcal{B}\delta\mathbf{u}^{(k)} = \mathbf{0}, \tag{3.44}$$

until convergence criteria $\|\delta\mathbf{u}^{(k)}\| \leq \epsilon$ is met for a chosen tolerance ϵ, where $\delta\mathbf{u}^{(k)}$ represents the Chebyshev coefficients of $\delta u^{(k)}$. If the criterion is not satisfied, we update $u^{(k+1)} = u^{(k)} + \delta u^{(k)}$ and continue iterating. Upon convergence at iteration $k = K$, the solution is represented by the final Chebyshev coefficient vector $\mathbf{u} = \mathbf{u}^{(K)}$, and the solution itself is approximated using (3.20).

3.5 Numerical examples

To evaluate the efficiency and accuracy of the ultraspherical spectral method, we examine four benchmark problems: two linear and two nonlinear ODEs. In each category, one problem has a known solution, while the other does not. For each problem, we also present the `Julia` code used.

For problems with explicit solutions, the convergence rate is quantified by the ℓ^2-error between the numerical and the true Chebyshev coefficients. Let $\mathbf{u}_{\text{exact}} \in \mathbb{R}^N$ denote the coefficients of the exact solution truncated at degree $N-1$, where N is chosen so that the $(N+1)$th coefficient is of machine precision, and $\mathbf{u}_{\text{num}} \in \mathbb{R}^n$ the coefficients of the numerical solution at approximation degree $n-1$. We construct an extended coefficient vector $\tilde{\mathbf{u}}_{\text{num}} = [\mathbf{u}_{\text{num}}, 0, \cdots, 0]^\intercal$ by zero-padding, where the number of zeros is $N - n$, and compute the error:

$$E_{\text{Explicit}}(n) = \left\|\tilde{\mathbf{u}}_{\text{num}} - \mathbf{u}_{\text{exact}}\right\|_2, \tag{3.45}$$

which is proven to converge to 0 as $n \to \infty$ by Theorem 3.3.1. We also plot the pointwise absolute errors over the spatial domain between the numerical solution and the exact solution.

For problems without explicit solutions, we use the Cauchy error. Let $\mathbf{u}_{\text{num}}^n \in \mathbb{R}^n$ and $\mathbf{u}_{\text{num}}^{\lceil 1.01n \rceil} \in \mathbb{R}^{\lceil 1.01n \rceil}$ be the coefficients computed at degrees n and $\lceil 1.01n \rceil$, respectively. After zero-padding $\mathbf{u}_{\text{num}}^n$ to match the dimension of $\mathbf{u}_{\text{num}}^{\lceil 1.01n \rceil}$, i.e., $\tilde{\mathbf{u}}_{\text{num}}^n = [\mathbf{u}_{\text{num}}^n, 0, \cdots, 0]^\intercal$, where the number of zeros is $\lceil 1.01n \rceil - n$, the Cauchy error

is defined as:

$$E_{\text{Cauchy}}(n) = \left\| \tilde{\mathbf{u}}^{n}_{\text{num}} - \mathbf{u}^{\lceil 1.01n \rceil}_{\text{num}} \right\|_2 . \tag{3.46}$$

The convergence of $E_{\text{Cauchy}}(n) \to 0$ implies the convergence of the numerical coefficients to the coefficients of the true solution, as guaranteed by the stability of the Woodbury QR algorithm used to solve (3.34) and the lemmas in Section 3.3. We plot the pointwise absolute differences between the two computed solutions, one using n, and the other using $\lceil 1.01n \rceil$.

All problems in the following are constructed using the package `OperatorApproximation.jl` [TL24].

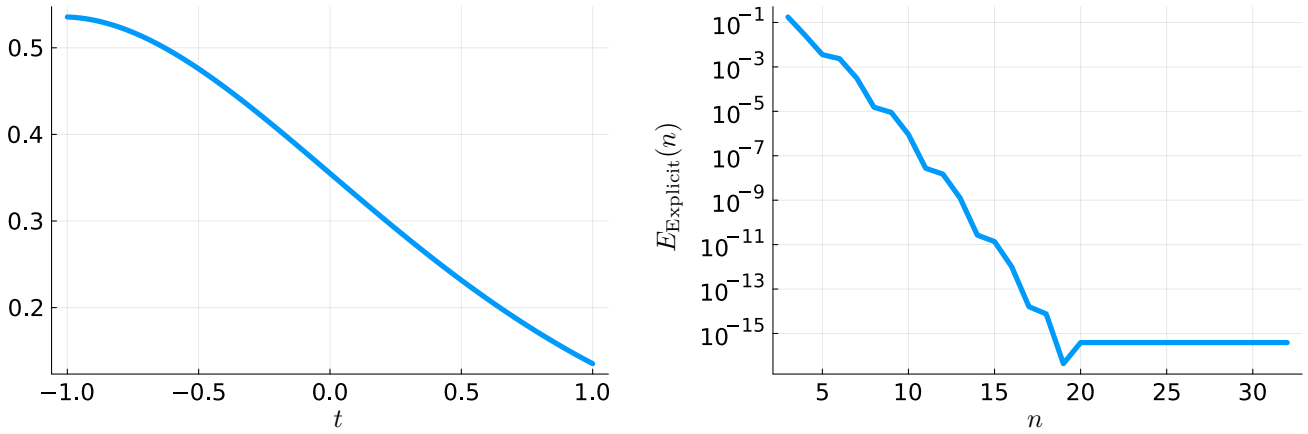
3.5.1 Linear problems

For our first linear problem with an explicit solution, we consider the Airy equation:

$$u''(t) - tu(t) = 0, \quad u(-1) = \text{Ai}(-1), \quad u'(-1) = \text{Ai}'(-1), \tag{3.47}$$

where $\text{Ai}(t)$ is the Airy function of the first kind. The exact solution is $u(t) = \text{Ai}(t)$. The following `Julia` code constructs and solves the resulting linear system.

```
gd = UltraMappedInterval(-1, 1, 0.0); sp = Ultraspherical(0, gd); sp2 = Ultraspherical(2, gd)
M = Multiplication(t -> -t); C = Conversion(sp2)
B1 = Matrix(Conversion(FixedGridValues([-1], gd))*sp, 1, n)
B2 = Matrix(Conversion(FixedGridValues([-1], gd))*Derivative()*sp, 1, n)
D = Matrix(Derivative(2)*sp, n - 2, n); A = Matrix(C*M*sp, n - 2, n); L = D + A
B = vcat(B1, B2, L); F = zeros(n, 1); F[1] = airyai(-1); F[2] = airyaiprime(-1)
bl, bu = bandwidths(L); bu = bl; B, Ub, V = generate_data2(L, n, vec(F), 2, bu)
X = woodbury_sparseqr(B, Ub, V, 2, bl)
```

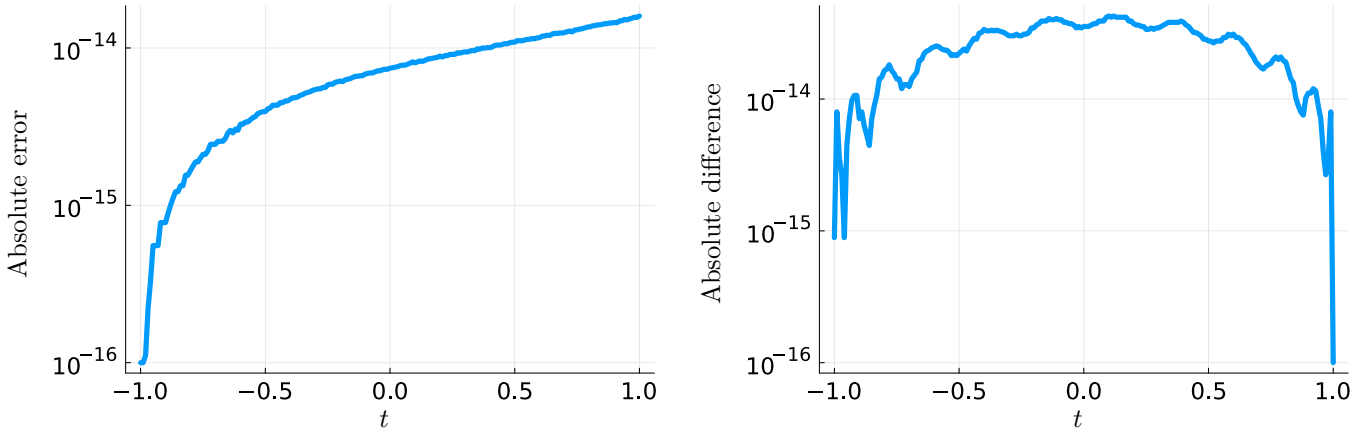


(a) The Airy function between -1 and 1

(b) The error for $N = 32$

Figure 3.1: (a) The Airy function $\text{Ai}(t)$ plotted over the domain $[-1, 1]$. (b) The error $E_{\text{Explicit}}(n)$, as defined in (3.45), decays to machine precision when $n \approx 18$.

In Figure 3.1(a), we plot the Airy function between -1 and 1, which is smooth. From Figure 3.1(b), we can see that the errors between the coefficients decay to machine precision when $n \approx 18$. For $n = 18$, we can see that the pointwise absolute errors over the spatial domain $[-1, 1]$ are basically of machine precision from Figure 3.2(a).



(a) Pointwise absolute errors for $n = 18$

(b) Pointwise absolute differences for $n = 24$

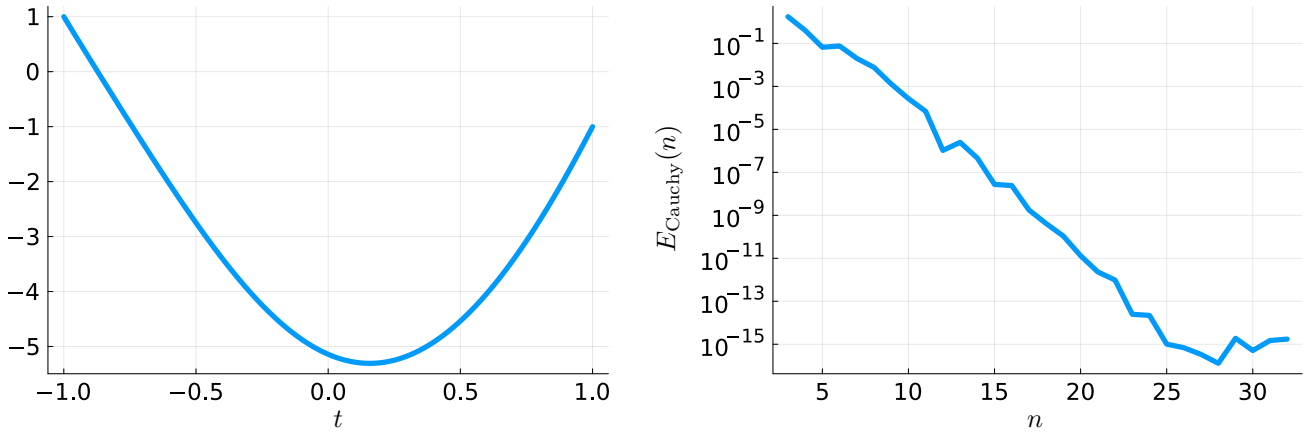
Figure 3.2: (a) Pointwise absolute errors for the Airy equation, evaluated at 201 uniform grid points between -1 and 1 using $n = 18$. (b) Pointwise absolute differences for the variable-coefficient ODE problem, evaluated at 201 uniform grid points between -1 and 1 using $n = 24$.

For our second linear problem without an explicit solution, we consider the following second-order ODE problem with variable coefficients:

$$u''(t) + (1 - t^2)u'(t) + (2 + 3t + t^3)u(t) = \cos(4t), \quad u(-1) = 1, \quad u(1) = -1. \tag{3.48}$$

The `Julia` code used to solve this problem is as follows.

```
gd = UltraMappedInterval(-1, 1, 0.0); sp = Ultraspherical(0, gd); sp2 = Ultraspherical(2, gd)
M1 = Multiplication(t -> 1 - t^2); M2 = Multiplication(t -> 2 + 3*t + t^3)
B1 = Matrix(Conversion(FixedGridValues([-1], gd))*sp, 1, n)
B2 = Matrix(Conversion(FixedGridValues([1], gd))*sp, 1, n)
C = Conversion(sp2); D = Matrix(Derivative(2)*sp, n - 2, n)
A = Matrix(C*M1*Derivative(1)*sp, n - 2, n); A2 = Matrix(C*M2*sp, n - 2, n)
B = D + A + A2; L = vcat(B1, B2, B)
f = BasisExpansion(t -> cos(4*t), sp, n - 2); F = vcat(1, -1, f.c)
bl, bu = bandwidths(L); bu = bl; B, Ub, V = generate_data2(L, n, vec(F), 2, bu)
X = woodbury_sparseqr(B, Ub, V, 2, bl)
```



(a) The numerical solution between -1 and 1

(b) The Cauchy error

Figure 3.3: (a) The numerical solution with $n = 32$. (b) The Cauchy error $E_{\text{Cauchy}}(n)$, as defined in (3.46), decays to machine precision at $n \approx 25$.

In Figure 3.3(a), we plot the numerical solution computed with $n = 32$. In Figure 3.3(b), we plot the Cauchy error $E_{\text{Cauchy}}(n)$, as defined in (3.46), comparing the numerical coefficients at polynomial degrees n and $\lceil 1.01n \rceil$. We can see that the Cauchy error decays to machine precision around $n = 25$. This result is expected since the solution is smooth over the relatively short spatial domain and the condition number of the linear system is relatively small. For example, with $n = 32$, the condition number of the almost banded matrix is $\mathcal{O}(10^3)$. Figure 3.2(b) shows the pointwise absolute differences between the two computed coefficients using $n = 24$ and $n = 25$. We can see that the difference is of machine precision.

3.5.2 Nonlinear problems

For our first nonlinear problem with an explicit solution, we consider a classical nonlinear ODE known as the logistic equation:

$$u'(t) - u(t) = -u^2(t), \quad u(-1) = \frac{1}{2}, \tag{3.49}$$

where the exact solution is given by $u(t) = 1/\left(1 + e^{-(t+1)}\right)$. For the nonlinear term $-u^2$ on the right-hand side of (3.49), we use Newton's method. Specifically, we first obtain an initial guess $u^{(0)}(t)$ by solving the simpler linear problem

$$u^{(0)'}(t) - u^{(0)}(t) = 0, \quad u^{(0)}(-1) = \frac{1}{2}. \tag{3.50}$$

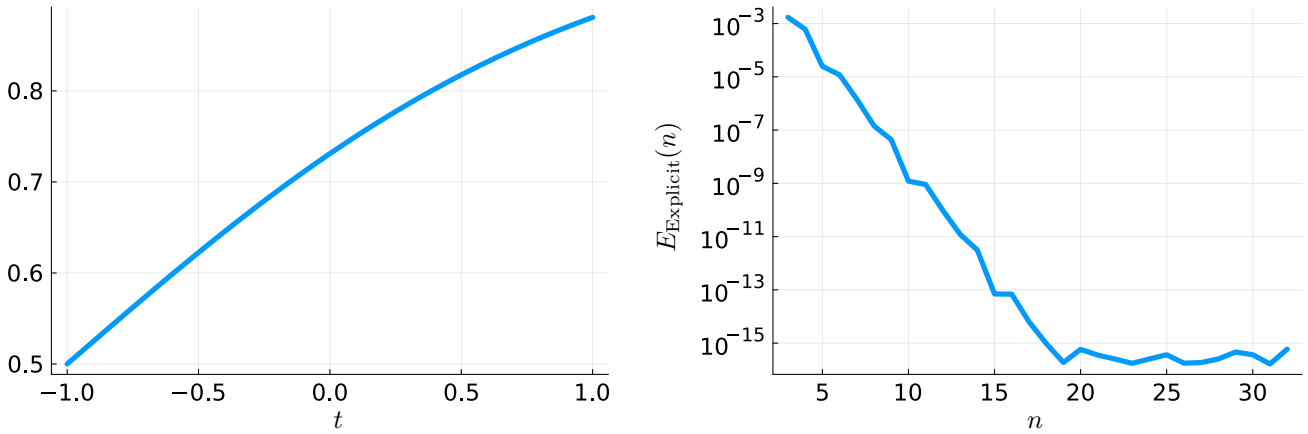
We then iteratively solve for the increments $\delta u^{(k)}$, $k = 0, 1, 2, \ldots$, satisfying the linearized equation

$$\delta u^{(k)'}(t) - \delta u^{(k)}(t) + 2u^{(k)}(t)\delta u^{(k)}(t) = -u^{(k)'}(t) + u^{(k)}(t) - u^{(k)2}(t), \quad \delta u^{(k)}(-1) = 0, \tag{3.51}$$

until the convergence criterion $\|\delta \mathbf{u}^{(k)}\| \leq 10^{-14}$ is achieved. The `Julia` code for this problem is as follows.

```
gd = UltraMappedInterval(-1, 1, 0.0); sp = Ultraspherical(0, gd); sp1 = Ultraspherical(1, gd)
B1 = Matrix(Conversion(FixedGridValues([-1], gd))*sp, 1, n)
D = Matrix(Derivative(1)*sp, n - 1, n); A = Matrix(Conversion(sp1)*sp, n - 1, n); BB = D - A
D2 = Matrix(Derivative(1)*sp, n, n); A2 = Matrix(Conversion(sp1)*sp, n, n); B2 = D2 - A2
L = vcat(B1, BB)
f = zeros(n - 1, 1); F = vcat(1/2, f)
bl, bu = bandwidths(L); bu = bl; B, Ub, V = generate_data2(L, n, vec(F), 1, bu)
u = woodbury_sparseqr(B, Ub, V, 1, bl)
for i = 1:20
    fu = BasisExpansion(sp, vec(Matrix(u))); uf = BasisExpansion(x -> -(fu(x))^2, sp, n)
    C = Matrix(Conversion(sp1)*Multiplication(x -> -2*fu(x))*sp, n - 1, n)
    L2 = vcat(B1, B - C); f2 = (-B2*u + A2*uf.c)[1:n - 1]; F = vcat(0, f2)
    bl, bu = bandwidths(L2); bu = bl; B, Ub, V = generate_data2(L2, n, vec(F), 1, bu)
    un = woodbury_sparseqr(B, Ub, V, 1, bl); res = norm(un); u = u + un
    if res < 1e-14
        break
    end
end
```

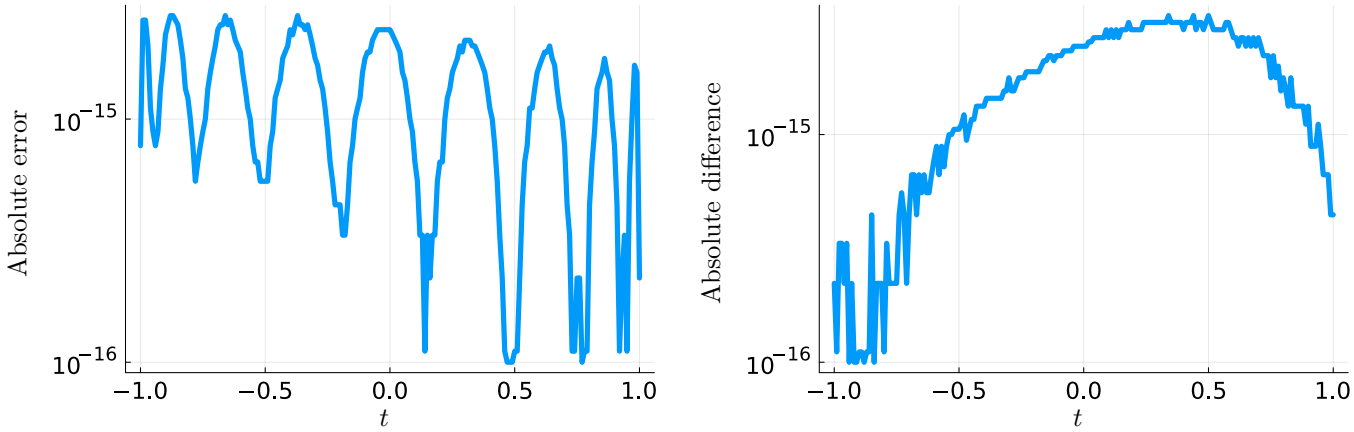
In Figure 3.7(a), we present the semilog plot of the residuals for solving (3.49) with $n = 32$, alongside a reference line that decays quadratically. We can see that 6 iterations are required to meet the convergence criterion, and from iteration 4 to iteration 6, the residuals decay quadratically. Figure 3.4(b) shows that the error $E_{\text{Explicit}}(n)$ rapidly decays to machine precision at $n \approx 18$ due to the smoothness of the logistic equation over the relatively short spatial domain. Figure 3.5(a) shows the pointwise absolute errors over the spatial domain $[-1, 1]$ using 201 uniform grid points are of machine precision.



(a) The logistic function between -1 and 1

(b) The error for $N = 32$

Figure 3.4: (a) The logistic function plotted over the domain $[-1, 1]$. (b) The error $E_{\text{Explicit}}(n)$ decays to machine precision at $n \approx 18$.



(a) Pointwise absolute errors for $n = 18$

(b) Pointwise absolute differences for $n = 36$

Figure 3.5: (a) Pointwise absolute errors for the logistic equation, evaluated at 201 uniform grid points between -1 and 1 using $n = 18$. (b) Pointwise absolute differences for the Duffing oscillator, evaluated at 201 uniform grid points between -1 and 1 using $n = 36$.

Next, we examine a more strongly nonlinear ODE known as the Duffing oscillator:

$$u''(t) + 0.1u'(t) - u(t) = 0.5\cos(2t) - u^3(t), \quad u(-1) = 0, \quad u'(-1) = 1. \tag{3.52}$$

To apply Newton's iteration, we first obtain an initial guess $u^{(0)}(t)$ by solving the simpler linear problem

$$u^{(0)''}(t) + 0.1u^{(0)'}(t) - u^{(0)}(t) = 0.5\cos(2t), \quad u^{(0)}(-1) = 0, \quad u^{(0)'}(-1) = 1. \tag{3.53}$$

We then iteratively solve for the increments $\delta u^{(k)}$, $k = 0, 1, 2, \ldots$, satisfying the linearized equation

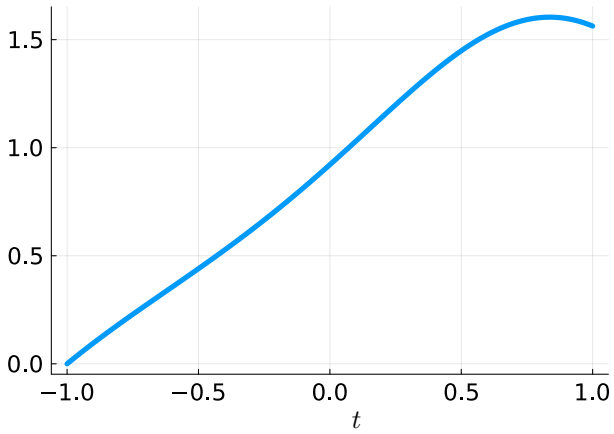
$$\delta u^{(k)''}(t) + 0.1\delta u^{(k)'}(t) - \delta u^{(k)}(t) + 3u^{(k)2}\delta u^{(k)}(t) = -u^{(k)''}(t) - 0.1u^{(k)'}(t) + u^{(k)}(t) + 0.5\cos(2t) - u^{(k)3}(t), \tag{3.54}$$

subject to $\delta u^{(k)}(-1) = 0$ and $\delta u^{(k)'}(-1) = 0$ until the convergence criterion $\|\delta \mathbf{u}^{(k)}\| \leq 10^{-14}$ is achieved.

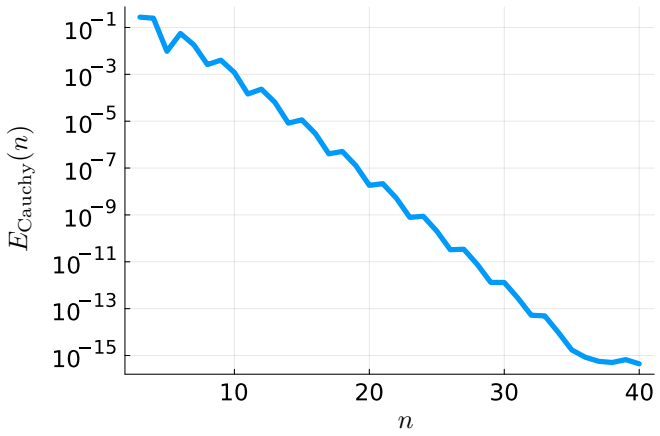
```
gd = UltraMappedInterval(-1, 1, 0.0); sp = Ultraspherical(0, gd); sp2 = Ultraspherical(2, gd)
B1 = Matrix(Conversion(FixedGridValues([-1], gd))*sp, 1, n)
B2 = Matrix(Conversion(FixedGridValues([-1], gd))*Derivative()*sp, 1, n)
D = Matrix(Derivative(2)*sp, n - 2, n); A = 0.1*Matrix(Conversion(sp2)*Derivative(1)*sp, n - 2, n)
A1 = Matrix(Conversion(sp2)*sp, n - 2, n); B = D + A - A1; D2 = Matrix(Derivative(2)*sp, n, n)
A2 = 0.1*Matrix(Conversion(sp2)*Derivative(1)*sp, n, n); A3 = Matrix(Conversion(sp2)*sp, n, n)
C = D2 + A2 - A3; L = vcat(B1, B2, B)
u0 = BasisExpansion(t -> 0.5*cos(2*t), sp, n-2); F = vcat(0, 1, u0.c)
bl, bu = bandwidths(L); bu = bl; B3, Ub, V = generate_data2(L, n, vec(F), 2, bu)
u = woodbury_sparseqr(B3, Ub, V, 2, bl)
for i = 1:20
    fu = BasisExpansion(sp,vec(Matrix(u))); uf = BasisExpansion(x -> -(fu(x))^3, sp, n)
    C1 = Matrix(Conversion(sp2)*Multiplication(x -> -3*(fu(x)^2))*sp, n - 2, n)
    L2 = vcat(B1, B2, B - C1); f2 = (-C*u + A3*uf.c)[1:n-2]; F = vcat(0,0,u0.c + f2)
    bl, bu = bandwidths(L2); bu = bl; B4, Ub, V = generate_data2(L2, n, vec(F), 2, bu)
    un = woodbury_sparseqr(B4, Ub, V, 2, bl); res = norm(un); u = u + un
    if res < 1e-14
        break
    end
end
```

In Figure 3.7(b), we plot the residuals from solving (3.54) iteratively with $n = 37$, alongside a reference line who decays quadratically. We can see that the residuals decay quadratically from the first iteration all the way to the end, and it requires 5 iterations in total to meet the convergence criterion.

Figure 3.6(b) shows that the Cauchy error $E_{\text{Cauchy}}(n)$ decays to machine precision near $n = 37$, which is still a small amount. With $n = 36$, we plot the pointwise absolute differences between two computed numerical solutions, one using $n = 36$ and the other using $n = 37$ as in Figure 3.5(b). The result shows that the difference is of machine precision.

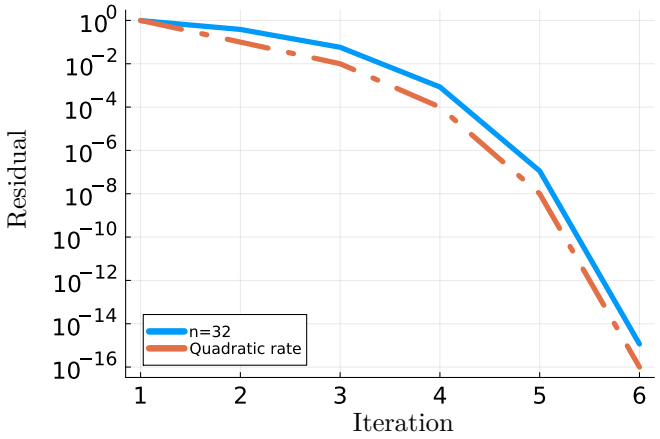


(a) The numerical solution between -1 and 1

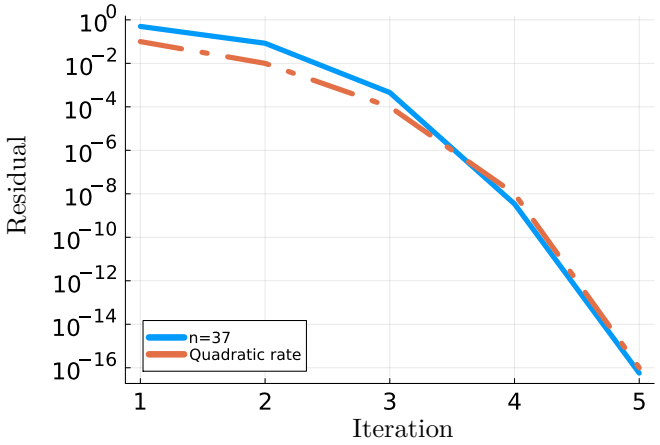


(b) The Cauchy error

Figure 3.6: (a) The numerical solution with $n = 42$. (b) The Cauchy error $E_{\text{Cauchy}}(n)$drops to machine precision at $n \approx 37$.



(a) Residuals from solving (3.51) iteratively



(b) Residuals from solving (3.54) iteratively

Figure 3.7: (a) Semilog plot of the residual versus iteration count using $n = 32$ and $\epsilon = 10^{-14}$ (b) Semilog plot of the residual versus iteration count using $n = 37$ and $\epsilon = 10^{-14}$.

Chapter 4

Spectral-in-time methods for solving linear time-evolution PDEs

Having introduced the ultraspherical spectral method for numerically solving linear and nonlinear ODEs [OT13], we now extend this methodology to time-evolution PDEs using spectral-in-time discretizations. As a preliminary step, this chapter restricts attention to linear PDEs, providing foundational insights essential for addressing the more general nonlinear scenarios discussed in the subsequent chapter.

Following the notations of [TO15], we consider a general class of linear PDEs with variable coefficients defined in a spatial-temporal domain $U_x \times U_t$, where U_x or U_t may be bounded or unbounded. Specifically, we study PDEs of the form:

$$\mathcal{L}u(x,t) = f(x,t), \quad \text{where} \quad \mathcal{L} = \sum_{i=0}^{N_t}\sum_{j=0}^{N_x} l_{ij}(x,t)\frac{\partial^{i+j}u}{\partial t^i \partial x^j}, \tag{4.1}$$

where $\mathcal{L}$ denotes a linear partial differential operator (PDO), and $N_x, N_t \in \mathbb{N}$ represent the highest orders of differentiation in the spatial (x) and temporal (t) directions, respectively. Throughout this and the subsequent chapter, we limit ourselves to the special case $N_t = 1$. The coefficient functions $l_{ij}(x,t)$ and the source term $f(x,t)$ are assumed to be sufficiently smooth on the domain $U_x \times U_t$. In alignment with Remark 2, we assume that the variable coefficients are polynomials in x and t, ensuring that the bandwidths of associated multiplication matrices remain constant with increasing problem dimension. The primary objective is to compute a solution satisfying the PDE (4.1) alongside linear boundary constraints:

$$\mathcal{B}_x u(x,t) = \mathbf{g}(t), \quad \mathcal{B}_t u(x,t) = \mathbf{h}(x), \tag{4.2}$$

where $\mathbf{g}(t) \in \mathbb{C}^{K_x}$ and $\mathbf{h}(x) \in \mathbb{C}^{K_t}$ are vector-valued boundary functions, and the operators $\mathcal{B}_x$, $\mathcal{B}_t$ act linearly on bivariate functions.

Before detailing the spectral-in-time approach, we briefly revisit spectral-in-space time-stepping strategies using either RK methods or LMMs. When either of the two is applied to PDEs of the form (4.1) with constraints (4.2), the spatial dependence of the solution $u(x,t)$ is approximated by selecting appropriate basis

functions that explicitly satisfy the spatial constraints $\mathcal{B}_x u(x,t) = \mathbf{g}(t)$. Substituting this spatial approximation into the PDE reduces it to a system of ODEs that govern the temporal evolution of the basis-function coefficients. The resulting ODE system can then be solved by standard time-stepping schemes.

Explicitly, we approximate $u(x,t)$ using a chosen set of spatial basis functions:

$$u(x,t) \approx \sum_{n=0}^{n_x-1} a_n(t) B_n(x), \tag{4.3}$$

where the spatial domain U_x contains the support of the basis functions $\{B_n\}_{n=0}^{n_x-1}$. Substituting (4.3) into (4.1) yields a first-order system of ODEs:

$$A_1 \mathbf{a}'(t) = A_0 \mathbf{a}(t) + A_1 \mathbf{b}(t), \tag{4.4}$$

assuming no mixed derivatives of $u(x,t)$ and the coefficient function of u_t is 1. Here, $\mathbf{a}(t) = [a_0(t), \ldots, a_{n_x-1}(t)]^\intercal$ and $\mathbf{b}(t) = [b_0(t), \ldots, b_{n_x-1}(t)]^\intercal$, where the forcing term is approximated as:

$$f(x,t) \approx \sum_{n=0}^{n_x-1} b_n(t) B_n(x). \tag{4.5}$$

Specifically, when $B_n = T_n$ (Chebyshev polynomials of the first kind), A_1 is a conversion matrix with bandwidth $\mathcal{O}(1)$ and A_0, resulting from the combinations of differentiation and multiplication matrices as detailed in Section 3.3, also has bandwidth $\mathcal{O}(1)$. Furthermore, this sparse structure of A_0 and A_1 remains valid for the other two sets of basis functions discussed later in Sections 4.3.2 and 4.3.3. Here, the notation $\mathcal{O}(1)$ denotes bandwidths that do not increase with the dimension of the problem.

To solve (4.4) while enforcing the temporal linear constraint $B_x \mathbf{a}(t) = \mathbf{g}(t)$, with $B_x \in \mathbb{C}^{K_x \times n_x}$, one approach involves building a sparse rectangular matrix $X \in \mathbb{C}^{n_x \times (n_x - K_x)}$ of full rank with a bandwidth $\mathcal{O}(K_x)$ such that $\mathbf{a}(t) = X\hat{\mathbf{a}}(t)$. When $\mathbf{g}(t) = \mathbf{0}$, it suffices to ensure $B_x X = \mathbf{0}$, thereby guaranteeing that $B_x \mathbf{a}(t) = B_x X \hat{\mathbf{a}}(t) = \mathbf{0}$.

If $\mathbf{g}(t) \neq \mathbf{0}$, we represent $\mathbf{a}(t)$ through the decomposition $\mathbf{a}(t) = X\hat{\mathbf{a}}(t) + \mathbf{a}_p$, where $\mathbf{a}_p = B_x^\intercal (B_x B_x^\intercal)^{-1} \mathbf{g}(t)$ is specifically chosen to satisfy $B_x \mathbf{a}_p = \mathbf{g}(t)$. This decomposition allows us to retain the previously constructed matrix X, satisfying $B_x X = 0$, when $\mathbf{g}(t) = \mathbf{0}$, thus ensuring that the boundary conditions remain consistently satisfied. Substituting this representation of $\mathbf{a}(t)$ into (4.4) yields:

$$A_1 X \hat{\mathbf{a}}'(t) = A_0 X \hat{\mathbf{a}}(t) + A_1 \mathbf{b}(t) + A_0 \mathbf{a}_p. \tag{4.6}$$

Since A_1 is invertible and X has full rank, multiplying both sides of (4.6) from the left by $X^\intercal$ leads to an invertible

matrix $X^\intercal A_1 X$. Consequently, we obtain the reduced system:

$$X^\intercal A_1 X \hat{\mathbf{a}}'(t) = X^\intercal A_0 X \hat{\mathbf{a}}(t) + X^\intercal A_1 \mathbf{b}(t) + X^\intercal A_0 \mathbf{a}_p. \tag{4.7}$$

Now, we solve (4.7) using the following two time-stepping methods and compute the corresponding computational complexities.

4.1 Multistage methods

RK methods, previously reviewed in Section 3.1, are well-established approaches for solving IVPs. Unlike conventional time-stepping methods that evolve discrete function values directly, we now evolve the coefficient vectors in time. To mitigate potential stability issues, our analysis focuses exclusively on DIRK and IRK methods.

We first consider a p-th order A-stable DIRK method with $s(p)$ stages. The computational complexity of solving (4.7) at each time step is dominated by scalar-vector multiplications and vector additions, resulting in $\mathcal{O}(s^2(p)n_x)$ operations per time step. Given a total of $N = T/\Delta t$ time steps, the overall computational complexity scales as $\mathcal{O}(Ns^2(p)n_x)$.

Alternatively, applying a p-th order A-stable IRK method with $s(p)$ stages involves solving a linear system at each time step characterized by a matrix A of dimension $s(p)n_x \times s(p)n_x$. The structure of this matrix is given by:

$$A_{i,j} = \begin{cases} d_i, & \text{if } i = j, \\ a_{i,j}, & \text{if } i \neq j \text{ and } |i-j| \equiv 0 \pmod 2, \\ 0, & \text{otherwise,} \end{cases} \tag{4.8}$$

where d_i represents diagonal entries and $a_{i,j}$ corresponds to non-zero off-diagonal elements occurring when $|i-j|$ is even. Without specialized algorithms exploiting this particular sparsity pattern, solving this system naively requires $\mathcal{O}(s^3(p)n_x^3)$ FLOPs per time step, since A is effectively dense. Consequently, with $N = T/\Delta t$ total time steps, the overall computational complexity is $\mathcal{O}(Ns^3(p)n_x^3)$.

Remark 3. *For a comprehensive examination of the relationship between the number of stages $s(p)$ and the order p of ERK methods, refer to [Wer80].*

4.2 Linear multistep methods

LMMs, previously reviewed in Section 3.2, represent another class of time-stepping techniques commonly used for solving IVPs. Due to Dahlquist's second barrier theorem [Dah63], explicit LMMs cannot achieve A-stability, and the highest achievable order of implicit A-stable LMMs is restricted to two (specifically, the trapezoidal method). To mitigate stability concerns, one may either reduce the time-step size Δt sufficiently to ensure stable performance of implicit LMMs or resort to BDF methods, provided the eigenvalues are not excessively dispersed around the origin. However, as highlighted in [TZ24], high-order BDF methods can encounter convergence difficulties when the time-step size lies within specific problematic intervals.

The computational complexity associated with applying an r-step LMM for solving (4.7) is again dominated by scalar-vector multiplications and vector additions, resulting in $\mathcal{O}(rn_x)$ operations per time step. Consequently, with $N = T/\Delta t$ total steps, the overall computational complexity is $\mathcal{O}(Nrn_x)$.

4.3 Spectral-in-time methods

We now introduce spectral-in-time methods, extending the framework of [OT13, TO15]. The solution $u(x,t)$, defined over the spatial-temporal domain $U_x \times [-1,1]$, is approximated by a bivariate spectral expansion:

$$u(x,t) \approx \sum_{i=0}^{n_t-1} \sum_{j=0}^{n_x-1} X_{ij}\, T_i(t)\, B_j(x), \tag{4.9}$$

where the basis functions for the temporal domain are the Chebyshev polynomials of the first kind $\{T_i\}_{i=0}^{n_t-1}$. The basis functions $\{B_j\}_{j=0}^{n_x-1}$ for the spatial domain U_x are chosen depending on the given spatial boundary conditions. The key idea behind the spectral-in-time method involves decomposing the linear partial differential operator (PDO) $\mathcal{L}$ from (4.1) into a finite sum of tensor products of ordinary differential operators (ODOs):

$$\mathcal{L} = \sum_{j=1}^{k} \left(\mathcal{L}_j^t \otimes \mathcal{L}_j^x\right), \tag{4.10}$$

where k denotes the splitting rank of $\mathcal{L}$, defined as the minimal number ensuring the validity of this decomposition. This formulation enables discretizing the PDE into a generalized Sylvester-type matrix equation comprising k terms:

$$A_1 X C_1^\intercal + \cdots + A_k X C_k^\intercal = F, \tag{4.11}$$

where matrices $A_j \in \mathbb{C}^{n_t \times n_t}$ and $C_j \in \mathbb{C}^{n_x \times n_x}$ represent truncated spectral discretizations of the ODOs in (4.10), and F contains the truncated bivariate spectral coefficients of the forcing function $f(x,t)$.

To solve equation (4.11), we first impose the linear boundary constraints. In [TO15, Section 6], the boundary conditions are directly imposed on (4.11) to decrease the degrees of freedom, which is difficult to understand. Here, we introduce a more straightforward way of doing so, which is similar to how the boundary conditions are imposed in the ODE case. First, the constraints $\mathcal{B}_x u(x,t) = \mathbf{g}(t)$ and $\mathcal{B}_t u(x,t) = \mathbf{h}(x)$ are discretized into matrix forms $XB_x^\intercal = G^\intercal$ and $B_t X = H$, respectively. Specifically, we define:

1. $B_x \in \mathbb{C}^{K_x \times n_x}$ as the discretization of the operator $\mathcal{B}_x$,
2. $G \in \mathbb{C}^{K_x \times n_t}$ containing the first n_t expansion coefficients of each component of $\mathbf{g}(t)$,
3. $B_t \in \mathbb{C}^{K_t \times n_t}$ as the discretization of the operator $\mathcal{B}_t$, and
4. $H \in \mathbb{C}^{K_t \times n_x}$ containing the first n_x expansion coefficients of each component of $\mathbf{h}(x)$.

For consistency, the matrices must satisfy the compatibility condition:

$$HB_x^\intercal = B_t G^\intercal. \tag{4.12}$$

To impose the constraints, we rewrite equation (4.11) along with the discretized constraints in the compact form:

$$\tilde{A}_1 X \tilde{C}_1^\intercal + \cdots + \tilde{A}_k X \tilde{C}_k^\intercal = \tilde{F}, \tag{4.13}$$

where, for each $i = 1, \ldots, k$,

$$\tilde{A}_i = \begin{pmatrix} \mathbf{0} \\ \hat{A}_i \end{pmatrix}, \quad \tilde{C}_i = \begin{pmatrix} \mathbf{0} \\ \hat{C}_i \end{pmatrix}, \quad \tilde{F} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{F} \end{pmatrix}, \tag{4.14}$$

and augmented constraint matrices are given by:

$$\tilde{B}_x = \begin{pmatrix} B_x \\ \mathbf{0} \end{pmatrix}, \quad \tilde{G} = \begin{pmatrix} G \\ \mathbf{0} \end{pmatrix}, \quad \tilde{B}_t = \begin{pmatrix} B_t \\ \mathbf{0} \end{pmatrix}, \quad \tilde{H} = \begin{pmatrix} H \\ \mathbf{0} \end{pmatrix}. \tag{4.15}$$

Here, $\hat{A}_i$ contains the first $n_t - K_t$ rows of A_i, $\hat{C}_i$ contains the first $n_x - K_x$ rows of C_i, and $\hat{F}$ is the principal $(n_t - K_t) \times (n_x - K_x)$ submatrix of F. Combining (4.13) with the augmented constraints yields:

$$X\tilde{B}_x^\intercal + \tilde{B}_t X + \sum_{i=1}^{k} \tilde{A}_i X \tilde{C}_i^\intercal = \tilde{G}^\intercal + \tilde{H} + \tilde{F}, \tag{4.16}$$

which, expressed using Kronecker products, is equivalent to:

$$\left[\tilde{B}_x \otimes I + I \otimes \tilde{B}_t + \sum_{j=1}^{k} \left(\tilde{C}_j \otimes \tilde{A}_j\right)\right] \mathrm{vec}(X) = \mathrm{vec}\left(\tilde{G}^\intercal + \tilde{H} + \tilde{F}\right). \tag{4.17}$$

Since (4.17) is actually derived by enforcing the sum of the linear constraints, one must verify that any solution to (4.17) indeed satisfies the constraints individually, namely, $X\tilde{B}_x^\intercal = \tilde{G}^\intercal$ and $\tilde{B}_t X = \tilde{H}$. To verify this, we first decompose $\tilde{B}_t$, $\tilde{B}_x$, $\tilde{G}$, $\tilde{H}$, and X as follows:

$$\tilde{B}_t = \begin{pmatrix} B_{1t} & B_{2t} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \quad \tilde{B}_x = \begin{pmatrix} B_{1x} & B_{2x} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \quad \tilde{G} = \begin{pmatrix} F_t^\intercal & F_{21}^\intercal \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \quad \tilde{H} = \begin{pmatrix} F_x & F_{12} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \quad X = \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix}, \tag{4.18}$$

where B_{1t} is the $K_t \times K_t$ principal submatrix of B_t, B_{1x} is the $K_x \times K_x$ principal submatrix of B_x, $F_t \in \mathbb{C}^{K_t \times K_x}$, $F_x \in \mathbb{C}^{K_t \times K_x}$, and $X_{11} \in \mathbb{C}^{K_t \times K_x}$ is the principal submatrix of X.

Proposition 1. *Suppose $B_{1x} \otimes I + I \otimes B_{1t}$ is invertible and the compatibility condition (4.12) is satisfied. Then any solution X to (4.17) also satisfies the boundary conditions.*

Proof. The compatibility condition (4.12) can be expressed as

$$B_{1t}F_t + B_{2t}F_{21} = F_x B_{1x}^\intercal + F_{12}B_{2x}^\intercal. \tag{4.19}$$

Using the boundary conditions $X\tilde{B}_x^\intercal = \tilde{G}^\intercal$ and $\tilde{B}_t X = \tilde{H}$, we have

$$B_{1t}X_{12} + B_{2t}X_{22} = F_{12}, \quad X_{21}B_{1x}^\intercal + X_{22}B_{2x}^\intercal = F_{21}. \tag{4.20}$$

Substituting (4.20) into (4.19) yields

$$B_{1t}(F_t - X_{12}B_{2x}^\intercal) = (F_x - B_{2t}X_{21})B_{1x}^\intercal. \tag{4.21}$$

The combined boundary condition $X\tilde{B}_x^\intercal + \tilde{B}_t X = \tilde{G}^\intercal + \tilde{H}$ further implies

$$B_{1t}X_{11} + B_{2t}X_{21} + X_{11}B_{1x}^\intercal + X_{12}B_{2x}^\intercal = F_x + F_t. \tag{4.22}$$

It suffices to show $B_{1t}X_{11} + B_{2t}X_{21} = F_x$ and $X_{11}B_{1x}^\intercal + X_{12}B_{2x}^\intercal = F_t$. Combining (4.22) and (4.21), after rearrangement, leads to

$$(B_{1x} \otimes I + I \otimes B_{1t})\,\mathrm{vec}(F_t - X_{12}B_{2x}^\intercal - X_{11}B_{1x}^\intercal) = 0. \tag{4.23}$$

Since $B_{1x}\otimes I + I \otimes B_{1t}$ is invertible, it follows that $F_t = X_{12}B_{2x}^\intercal + X_{11}B_{1x}^\intercal$, thus confirming that the boundary conditions are satisfied. □

In the next three sections, we introduce two additional sets of basis functions beyond Chebyshev polynomials to accommodate various boundary condition scenarios.

4.3.1 Finite spatial domain with non-periodic boundary conditions

For the spatial-temporal domain $[a, b] \times [t_0, t_M]$ with nonperiodic spatial boundary conditions, we may also select $B_j(x) = T_j(x)$ for $j = 0, \ldots, n_x - 1$. In fact, this is the ultraspherical spectral method [TO15]. In this case, the approximation of $u(x, t)$ is given by:

$$u(x,t) \approx \sum_{i=0}^{n_t-1}\sum_{j=0}^{n_x-1} X_{ij}\, T_i\left(\psi(t)\right) T_j\left(\phi(x)\right), \quad \phi(x) = \frac{2x-a-b}{b-a}, \quad \psi(t) = \frac{2t-t_0-t_M}{t_M-t_0}. \tag{4.24}$$

To construct equation (4.17), we must compute the associated differentiation, conversion, and multiplication matrices as detailed in Section 3.3.

Although we have limited our attention to polynomial-type coefficient functions, particular attention is required when encountering a coefficient function possessing an unbounded splitting rank. In such scenarios, the operator should be approximated by one with finite splitting rank. The exact separable approximation can be computed by [TO15, Proposition 4.2], alongside the tensor-train decomposition of functions described in [Ose11].

4.3.2 Finite spatial domain with periodic boundary conditions

For the same spatial-temporal domain $[a, b] \times [t_0, t_M]$ with periodic spatial boundary conditions, we choose the spatial basis functions as complex exponentials $B_j(x) = e^{ij\phi(x)}$ for $j = -M, \ldots, M$, where the coordinate

transformation $\phi(x)$ is defined by

$$\phi(x) = \frac{2\pi(x-a)}{b-a}. \tag{4.25}$$

Hence, the approximation of $u(x,t)$ is given by:

$$u(x,t) \approx \sum_{k=0}^{n_t-1} \sum_{j=-M}^{M} X_{k,j}\, T_k(\psi(t))\, e^{ij\phi(x)}, \quad \psi(t) = \frac{2t - t_0 - t_M}{t_M - t_0}. \tag{4.26}$$

Analogous to the ultraspherical spectral method, we first examine the kth-order differentiation matrix $\mathbf{D}_k$. Since $\phi'(x) = 2\pi/(b-a)$, the kth derivative of the basis function $e^{ij\phi(x)}$ becomes $(ij\phi'(x))^k e^{ij\phi(x)}$. Consequently, the differentiation matrix $\mathbf{D}_k$, expressed in the truncated Fourier basis $\{e^{ij\phi(x)}\}_{j=-M}^{M}$, is diagonal and takes the form:

$$\mathbf{D}_k = \operatorname{diag}\left((i(-M)\phi')^k,\, (i(-M+1)\phi')^k,\, \ldots,\, (iM\phi')^k\right). \tag{4.27}$$

To implement multiplication by a spatially periodic coefficient function $a^i(x)$, approximated by its Fourier expansion

$$a^i(x) \approx \sum_{m=-M}^{M} \hat{a}_m\, e^{im\phi(x)}, \tag{4.28}$$

we note that the product $a^i(x)u(x)$ corresponds to a discrete convolution in Fourier coefficient space. Specifically, given

$$u(x) \approx \sum_{j=-M}^{M} \hat{u}_j\, e^{ij\phi(x)}, \tag{4.29}$$

the Fourier coefficients $\hat{c}_k$ of the product $a^i(x)u(x)$ are computed via the convolution sum:

$$\hat{c}_k = \sum_{m=-M}^{M} \hat{a}_m\, \hat{u}_{k-m}. \tag{4.30}$$

This convolution operation corresponds to multiplication by a Toeplitz matrix, commonly denoted $\mathbf{M}[a^i]$, in the coefficient domain. Detailed discussions on Fourier-based multiplication operators and efficient convolution implementations are available in [Tre00, Boy01].

Unlike ultraspherical polynomials, where repeated differentiation transit coefficients from T_n to U_{n-1} or, more generally, between different families of $C_n^{(k)}$, exponential functions remain invariant under differentiation as eigenfunctions of the differentiation operator. Therefore, the Fourier basis does not require conversion matrices. All differentiation and multiplication operations act directly and naturally on $e^{ij\phi(x)}$, resulting in diagonal differenti-

ation matrices and Toeplitz-structured multiplication matrices. As a result, the resulting matrix is purely banded instead of being almost banded.

4.3.3 Unbounded spatial domain

For the spatial-temporal domain $\mathbb{R} \times [t_0, t_M]$ subject to spatial boundary conditions $u(x,t) \to 0$ as $x \to \pm\infty$, we employ rational basis functions defined explicitly by:

$$B_j(x) = R_j(x) = \left(\frac{x-i}{x+i}\right)^j, \tag{4.31}$$

which is a Möbius transform from the real line to the unit circle. The solution $u(x,t)$ is then approximated by the expansion:

$$u(x,t) \approx \sum_{k=0}^{n_t-1} \sum_{j=-M}^{M} X_{k,j}\, T_k(\psi(t))\, R_j(x), \quad \psi(t) = \frac{2t - t_0 - t_M}{t_M - t_0}. \tag{4.32}$$

As in previous subsections, we must construct two primary types of operators: the kth-order differentiation operator $\mathcal{D}_k$ and the multiplication operator $\mathcal{M}[a^i]$ corresponding to the coefficient function $a^i(x)$. Like the Fourier basis, the rational basis $\{R_j\}$ remains invariant under differentiation, thus eliminating the need for conversion operators. Explicitly, differentiating the rational function yields:

$$R_j'(x) = j\left(\frac{x-i}{x+i}\right)^{j-1} \frac{d}{dx}\left(\frac{x-i}{x+i}\right) = \frac{2ij}{(x+i)^2}\left(\frac{x-i}{x+i}\right)^{j-1}. \tag{4.33}$$

Using the identity

$$\frac{1}{(x+i)^2} = \frac{1}{2}\left(R_1(x) - 1\right) - \frac{1}{4}\left(R_2(x) - 1\right), \tag{4.34}$$

we derive the recurrence relation:

$$R_j'(x) = -\frac{ij}{2} R_{j-1}(x) + ij R_j(x) - \frac{ij}{2} R_{j+1}(x). \tag{4.35}$$

In matrix form, the first-order differentiation operator $\mathcal{D}_1$ is represented as:

$$\mathcal{D}_1 = \begin{pmatrix} \ddots & i & 0 & 0 & 0 & 0 & \cdots \\ \cdots & -2i & \frac{i}{2} & 0 & 0 & 0 & \cdots \\ \cdots & i & -i & 0 & 0 & 0 & \cdots \\ \cdots & 0 & \frac{i}{2} & 0 & -\frac{i}{2} & 0 & \cdots \\ \cdots & 0 & 0 & 0 & i & -i & \cdots \\ \cdots & 0 & 0 & 0 & -\frac{i}{2} & 2i & \cdots \\ \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \tag{4.36}$$

and higher-order differentiation operators are computed as $\mathcal{D}_k = \mathcal{D}_1^k$.

Regarding multiplication matrices, as we restrict ourselves to polynomial coefficient functions, these can be conveniently constructed from combinations of differentiation operators. For example, consider the operator $x\frac{d}{dx}$. Acting on the rational basis, we have:

$$xR_j'(x) = jR_{j-1}(x)\frac{2ix}{(x+i)^2}. \tag{4.37}$$

Expressing the rational factor as

$$\frac{2ix}{(x+i)^2} = c_0 + c_1\frac{x-i}{x+i} + c_2\left(\frac{x-i}{x+i}\right)^2, \tag{4.38}$$

and matching coefficients, we obtain $c_0 = 1/2$, $c_1 = 0$, and $c_2 = -1/2$. Thus, the operator is represented by:

$$xR_j'(x) = \frac{1}{2}jR_{j-1}(x) - \frac{1}{2}jR_{j+1}(x). \tag{4.39}$$

In matrix form, the operator $x\frac{d}{dx}$ is given explicitly by:

$$\mathcal{M}\left[x\right]\frac{d}{dx} = \begin{pmatrix} \ddots & -1 & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & -\frac{1}{2} & 0 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & 1 & \cdots \\ \cdots & 0 & 0 & 0 & -\frac{1}{2} & 0 & \cdots \\ \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{4.40}$$

Similarly, other polynomial-related operators can be constructed from $\mathcal{M}[x\frac{d}{dx}]$; for instance, the operator $x^2\frac{d^2}{dx^2}$ can be represented as:

$$\mathcal{M}\left[x^2\right]\frac{d^2}{dx^2} = \left(\mathcal{M}\left[x\right]\frac{d}{dx}\right)^2 - \mathcal{M}\left[x\right]\frac{d}{dx}. \tag{4.41}$$

Although polynomial-based operators do not capture every possible coefficient function, more general problems can still be effectively handled by truncating the spatial domain to a sufficiently large finite interval $[x_0, x_M]$. In practice, as the temporal domain increases (that is, as t_M grows), the spatial interval must expand accordingly, and suitable boundary conditions $u(x,t) = 0$ at $x = x_0$ and $x = x_M$ should be imposed.

Up to this point, we have introduced all basis functions for discretizing the spatial domain under various boundary conditions, alongside the Chebyshev polynomials employed for temporal discretization. Using these basis functions, the matrix on the left-hand side of equation (4.17) becomes an almost banded matrix of dimension $\mathcal{O}(n_x n_t) \times \mathcal{O}(n_x n_t)$. This matrix exhibits a bandwidth of $\mathcal{O}(n_t)$, except for its first $\mathcal{O}(n_t)$ rows, which remain dense [TO15].

The computational complexities associated with solving (4.7) and (4.17) using various discretization methods are summarized in Table 4.1. As demonstrated in the table, the computational complexity associated with applying

Matrix Size	**Bandwidth**	**Numerical Method**	**Complexity**
n_x	$\mathcal{O}(1)$	p-th order $s(p)$-stage DIRK (N steps)	$\mathcal{O}\left(Ns^2(p)n_x\right)$
		p-th order $s(p)$-stage IRK (N steps)	$\mathcal{O}\left(Ns^3(p)n_x^3\right)$
		r-step LMM (N steps)	$\mathcal{O}\left(Nrn_x\right)$
$n_x n_t$	$\mathcal{O}(n_t)$	Woodbury QR	$\mathcal{O}\left(n_t^3 n_x\right)$

Table 4.1: Computational complexities for solving (4.7) and (4.17) with various numerical methods.

the Woodbury QR algorithm to the almost banded linear system derived from the spectral-in-time discretization scales as $\mathcal{O}(n_t^3 n_x)$, indicating a cubic dependency on the temporal resolution. In practice, standard time-stepping methods, such as DIRK or LMM, generally offer superior efficiency compared to the spectral-in-time method when using moderate numbers of time steps. Of course, if the resulting $n_x n_t \times n_x n_t$ matrix is purely banded, then the complexity to solve such a linear system using the optimized QR decomposition (Algorithm 5) is $\mathcal{O}(n_x n_t)$.

One might initially suspect a trade-off between the spectral convergence provided by spectral-in-time methods and the reduced computational complexity typical of traditional methods such as RK methods or LMMs. However, it is indeed possible to achieve improved computational efficiency while retaining spectral convergence

by reformulating the spectral-in-time approach within a time-stepping framework.

Instead of solving the problem (4.17) in the entire domain as in the plot on the left of Figure 4.1, where n_x basis functions are used in the spatial domain $[a, b]$ and n_t basis functions are used in the entire temporal domain $[t_0, t_M]$, we partition the temporal domain into smaller intervals as in the plot on the right of Figure 4.1, where some smaller value of n_t is used in each subtemporal domain. Specifically, we divide $[t_0, t_M]$ into N subintervals, each of length $\Delta t = |t_M - t_0| / N$, and solve sequentially on each subdomain $[a, b] \times [t_0 + (k-1)\Delta t,\, t_0 + k\Delta t]$ for $k = 1, \ldots, N$. Upon solving each subproblem, we obtain the approximate coefficients at the subinterval

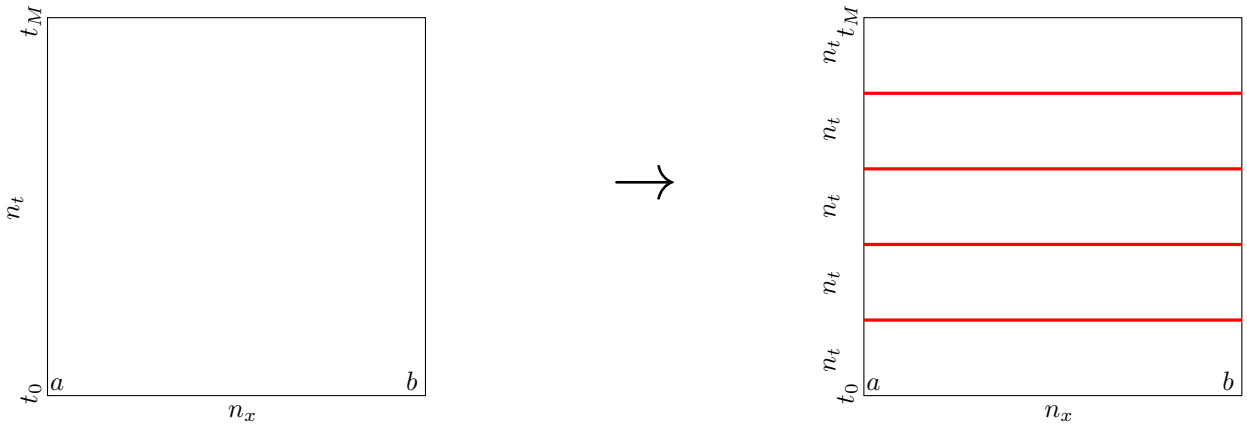


Figure 4.1: Comparison of one global solve and a time-stepping approach.

endpoint, which then serve as initial condition for the subsequent subinterval. Proceeding in this iterative manner, the solution coefficients are advanced step-by-step from the initial time t_0 to the final time t_M. This approach involves solving N subproblems, each with n_t temporal coefficients, where n_t is expected to be smaller than the one used in one global solve since by dividing the temporal domain, we hope using less temporal coefficients is good enough to achieve a certain level of accuracy in each subinterval. In practice, the value of n_t can be obtained by computing the difference at the final time point of each subproblem between one using n_t and another one using $\lceil 1.01 n_t \rceil$ with n_x being fixed. If the difference is small, then we know that this value of n_t should be good enough for this subproblem, and we can do this adaptively for other subproblems. Besides, we also have the following theorem to tell us the relation between n_t and Δt for a fixed value of n_x and ϵ:

Theorem 4.3.1 (Jackson's fourth theorem [Jac11, Jac13])**.** *Suppose $f \in C^{k,\alpha}([-1, 1])$ and $f^{(k)}$ has α-Hölder constant L. Then for $n > k$,*

$$\inf_{p \in \mathcal{P}_n} \|f - p\|_\infty \leq \frac{L \pi^{2(k+\alpha)}}{(4n+6)(4n+4)\cdots(4n-4k+10)(4n-4k+6)^\alpha}. \tag{4.42}$$

For $f \in C^{k,\alpha}([t_0, t_0 + \Delta t])$ with $f^{(k)}$ having α-Hölder constant L_f, we define the following transformation

functions:

$$u = \phi(t) = \frac{2t - (2t_0 + \Delta t)}{\Delta t}, \quad t = \phi^{-1}(u) = t_0 + \frac{\Delta t}{2}(u+1). \tag{4.43}$$

Therefore, the error in approximating f on $[t_0, t_0 + \Delta t]$ is the same as approximating g on $[-1, 1]$ with $g(u) = f(\phi^{-1}(u)) = f\left(t_0 + \frac{\Delta t}{2}(u+1)\right)$. The α-Hölder constant for $g^{(k)}$ can be derived as $L_g = L_f\left(\frac{\Delta t}{2}\right)^{k+\alpha}$, and by Theorem 4.3.1, we can get the following bound:

$$\|g - p\|_\infty \leq C_{k,\alpha} L_f \left(\frac{\Delta t}{n}\right)^{k+\alpha}, \quad C_{k,\alpha} = \left(\frac{\pi^2}{8}\right)^{k+\alpha}. \tag{4.44}$$

For a fixed level of precision ϵ, we want $C_{k,\alpha} L_f \left(\frac{\Delta t}{n}\right)^{k+\alpha} \leq \epsilon$, which implies $n \geq \Delta t \left(\frac{C_{k,\alpha} L_f}{\epsilon}\right)^{\frac{1}{k+\alpha}}$. Thus, for fixed n_x and ϵ, n_t scales linearly with Δt.

We refer to this strategy as the N-step spectral time-stepping method, and we will compare its overall computational complexity and the accuracy at $t = t_M$ against the traditional global spectral-in-time approach in the following numerical experiments.

4.4 Numerical examples

To assess the efficiency and precision of the spectral-in-time method, we examine six benchmark problems, two corresponding to each type of spatial basis function. Within each pair, one problem admits an explicit solution, while the other does not.

For problems with explicit solutions, the convergence rates are measured by the ℓ^2-error between the numerical and exact bivariate coefficients. Let $\mathbf{u}_{\text{exact}} \in \mathbb{R}^{\tilde{n}_t \times \tilde{n}_x}$ denote the coefficient matrix of the exact solution, truncated to sufficiently large degrees $\tilde{n}_t$ and $\tilde{n}_x$ so that the subsequent coefficients in both the spatial and temporal domains are numerically negligible. Let $\mathbf{u}_{\text{num}} \in \mathbb{R}^{n_t \times n_x}$ be the coefficient matrix of the numerical solution computed at the degree of approximation $n_t \times n_x$. To facilitate error computation, we zero-pad $\mathbf{u}_{\text{num}}$ into an extended matrix:

$$\tilde{\mathbf{u}}_{\text{num}} = \begin{pmatrix} \mathbf{u}_{\text{num}} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{\tilde{n}_t \times \tilde{n}_x}, \tag{4.45}$$

and compute the error as:

$$E_{\text{Explicit}}(n_t, n_x) = \left\|\tilde{\mathbf{u}}_{\text{num}} - \mathbf{u}_{\text{exact}}\right\|_2 . \tag{4.46}$$

If adding an additional term in either the temporal or spatial dimension significantly increases the numerical error, we can analyze the coefficient errors separately in each domain. Specifically, if an additional spatial term introduces substantial error, we examine the temporal coefficient error $E_{\text{Explicit}}(n_t, \tilde{n}_x)$. Similarly, if temporal refinement leads to substantial increased error, we examine the spatial coefficient error $E_{\text{Explicit}}(\tilde{n}_t, n_x)$.

To compare the N-step spectral time-stepping method with the global spectral-in-time approach, we compute the pointwise absolute error at the final time $t = t_M$:

$$\left|u_{\text{num}}(x_i, t_M) - u_{\text{exact}}(x_i, t_M)\right|, \quad i = 1, \ldots, K, \quad x_i \in U_x. \tag{4.47}$$

Here, $u_{\text{num}}(\cdot, t_M)$ denotes the numerical solution recovered from the computed coefficients at $t = t_M$ using a N-step spectral time-stepping method, while $u_{\text{exact}}(\cdot, t_M)$ represents the exact solution at $t = t_M$. For each subproblem, the temporal resolution is denoted by n_t, and the spatial resolution is fixed at $n_x = \tilde{n}_x$, matching that of $\mathbf{u}_{\text{exact}}$.

For problems without explicit solutions, or for which it is non-trivial to obtain the bivariate coefficient expansions of the exact solutions, the convergence is quantified using the Cauchy error. Let $\mathbf{u}_{\text{num}}^{n_t \times n_x} \in \mathbb{R}^{n_t \times n_x}$ and $\mathbf{u}_{\text{num}}^{\lceil 1.01 n_t \rceil \times \lceil 1.01 n_x \rceil} \in \mathbb{R}^{\lceil 1.01 n_t \rceil \times \lceil 1.01 n_x \rceil}$ be numerical solutions computed at successive approximation degrees. Again, we zero-pad the lower-degree solution as follows:

$$\tilde{\mathbf{u}}_{\text{num}}^{n_t \times n_x} = \begin{pmatrix} \mathbf{u}_{\text{num}}^{n_t \times n_x} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{\lceil 1.01 n_t \rceil \times \lceil 1.01 n_x \rceil}, \tag{4.48}$$

and define the Cauchy error as:

$$E_{\text{Cauchy}}(n_t, n_x) = \left\| \tilde{\mathbf{u}}_{\text{num}}^{n_t \times n_x} - \mathbf{u}_{\text{num}}^{\lceil 1.01 n_t \rceil \times \lceil 1.01 n_x \rceil} \right\|_2 . \tag{4.49}$$

If adding an additional term in either the temporal or spatial dimension significantly increases the numerical error, we can analyze the coefficient Cauchy errors separately in each domain. Specifically, if an additional spatial term

introduces substantial error, we examine the temporal coefficient Cauchy error defined by

$$E_{\text{Cauchy}}(n_t) = \left\| \tilde{\mathbf{u}}_{\text{num}}^{n_t \times n_x} - \mathbf{u}_{\text{num}}^{\lceil 1.01 n_t \rceil \times n_x} \right\|_2. \tag{4.50}$$

Similarly, if temporal refinement leads to substantial increased error, we examine the spatial coefficient Cauchy error

$$E_{\text{Cauchy}}(n_x) = \left\| \tilde{\mathbf{u}}_{\text{num}}^{n_t \times n_x} - \mathbf{u}_{\text{num}}^{n_t \times \lceil 1.01 n_x \rceil} \right\|_2. \tag{4.51}$$

To compare the N-step spectral time-stepping method and the global spectral-in-time approach for problems without explicit solutions, we again compute the pointwise absolute error at the final time $t = t_M$:

$$\left| u_{\text{num}}^{n_t \times \hat{n}_x}(x_i, t_M) - u_{\text{ref}}^{\hat{n}_t \times \hat{n}_x}(x_i, t_M) \right|, \quad i = 1, \ldots, K, \quad x_i \in U_x. \tag{4.52}$$

Here, $u_{\text{num}}^{n_t \times \hat{n}_x}(\cdot, t_M)$ denotes the numerical solution recovered from the computed coefficients at $t = t_M$ using a N-step spectral time-stepping method, while $u_{\text{ref}}^{\hat{n}_t \times \hat{n}_x}(\cdot, t_M)$ denotes a high-resolution reference solution at $t = t_M$. The degrees $\hat{n}_t$ and $\hat{n}_x$ are chosen sufficiently large to ensure that the reference solution closely approximates the true underlying solution. The temporal resolution per subproblem is again denoted by n_t, and the spatial resolution is fixed at $n_x = \hat{n}_x$.

4.4.1 Finite spatial domain with non-periodic boundary conditions

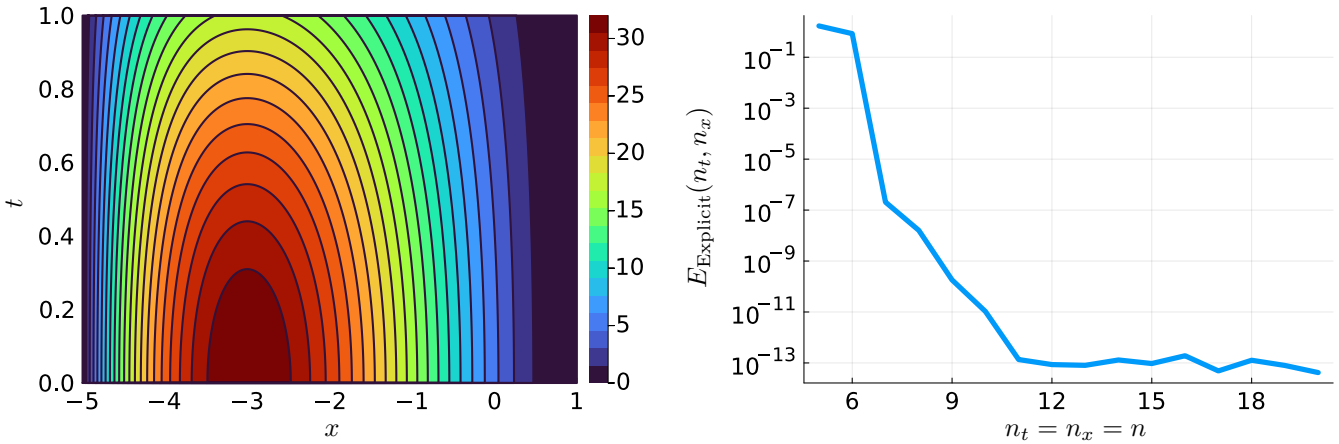
For our first problem with an explicit solution, we consider the following linearized Korteweg–de Vries (KdV) equation with a forcing term:

$$u_t + u_{xxx} = 6\cos(t) - (x+5)(1-x)^2 \sin(t), \quad (x,t) \in [-5,1] \times [0,1], \tag{4.53}$$

subject to boundary conditions $u(-5,t) = u(1,t) = u_x(1,t) = 0$ and the initial condition $u(x,0) = (x+5)(1-x)^2$. The exact solution is given by $u(x,t) = (x+5)(1-x)^2\cos(t)$. The `Julia` code used to solve this problem is as follows, where `t_0` $= 0$, `u0` is the coefficient vector of the initial condition, and `dt` $= 1/N$ for a N-step spectral time-stepping approach.

```
gd_t = UltraMappedInterval(t_0, t_0 + dt, 0.0); sp_t = Ultraspherical(0, gd_t)
sp1_t = Ultraspherical(1, gd_t)
```

```
gd_x = UltraMappedInterval(-5, 1, 0.0); sp_x = Ultraspherical(0, gd_x)
sp3_x = Ultraspherical(3, gd_x)
A1 = Matrix(Derivative() * sp_t, n_t-1, n_t); A2 = Matrix(Conversion(sp1_t)*sp_t, n_t-1, n_t)
C1 = Matrix(Conversion(sp3_x)*sp_x, n_x-3, n_x); C2 = Matrix(Derivative(3)*sp_x, n_x-3, n_x)
B_t = reshape([(-1)^(j+1) * sqrt(2) for j in 1:n_t],1,n_t); B_t[1] = 1
B_x1 = Matrix(Conversion(FixedGridValues([-5,1], gd_x)) * sp_x, 2, n_x)
B_x2 = Matrix(Conversion(FixedGridValues([1],gd_x))*Derivative()*sp_x,1,n_x)
Z1 = spzeros(1, n_t); Z2 = spzeros(3, n_x); Z3 = spzeros(n_t-1, n_t); Z4 = spzeros(n_x-3, n_x)
idx = sparse(I, n_x, n_x); idt = sparse(I, n_t, n_t)
AA1 = vcat(Z1,A1); AA2 = vcat(Z1,A2)
CC1 = vcat(Z2,C1); CC2 = vcat(Z2,C2)
BBt = vcat(B_t, Z3); BBx = vcat(B_x1,B_x2, Z4)
L = kron(idx, BBt) + kron(BBx, idt) + kron(CC1, AA1) + kron(CC2, AA2)
F = zeros(n_t,n_x)
c_n = BasisExpansion(t -> sin(t), sp1_t, n_t-1).c
c_k = BasisExpansion(x -> -(x+5)*(1-x)^2, sp3_x, n_x-3).c
c_n2 = BasisExpansion(t -> cos(t), sp1_t, n_t-1).c; c_k2 = BasisExpansion(x -> 6, sp3_x, n_x-3).c
F_2 = c_n2 * transpose(c_k2) + c_n * transpose(c_k); F[2:end,4:end] = F_2; F[1,:] = transpose(u0)
bl, bu = bandwidths(kron(CC1,AA1) + kron(CC2,AA2))
k = 3*n_t; B,Ub,V = generate_data2(L, n_t*n_x, vec(F), k, bu)
XX = woodbury_sparseqr(B, Ub, V, k, bl); X = reshape(XX, n_t, n_x)
```



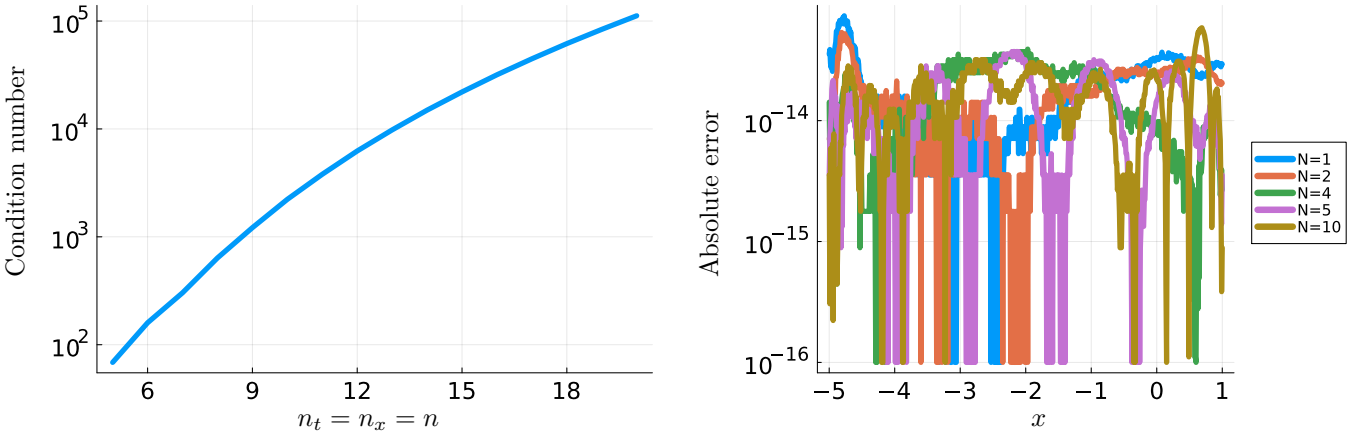
(a) Contour plot of $u(x,t)$

(b) Coefficient errors for $\tilde{n}_t = \tilde{n}_x = 20$

Figure 4.2: (a) Contour plot of the exact solution $u(x,t) = (x+5)(1-x)^2\cos(t)$ on $(x,t) \in [-5,1] \times [0,1]$. (b) The coefficient error $E_{\text{Explicit}}(n_t, n_x)$, defined as in (4.46), decays to $\mathcal{O}(10^{-13})$ at $n \approx 11$.

Figure 4.2(a) shows the contour plot of the exact solution $u(x,t)$, showing its polynomial behavior in space.

Figure 4.2(b) shows the coefficient error $E_{\text{Explicit}}(n_t, n_x)$ for the global solve, which decreases exponentially up to the polynomial degree $n \approx 11$, indicating the spectral convergence expected for a smooth solution. However, the error does not fully reach machine precision, plateauing instead around $\mathcal{O}(10^{-13})$. This phenomenon results from the increasing condition number of the discretization matrix, as depicted in Figure 4.3(a).



(a) Condition number of the matrix in (4.17)

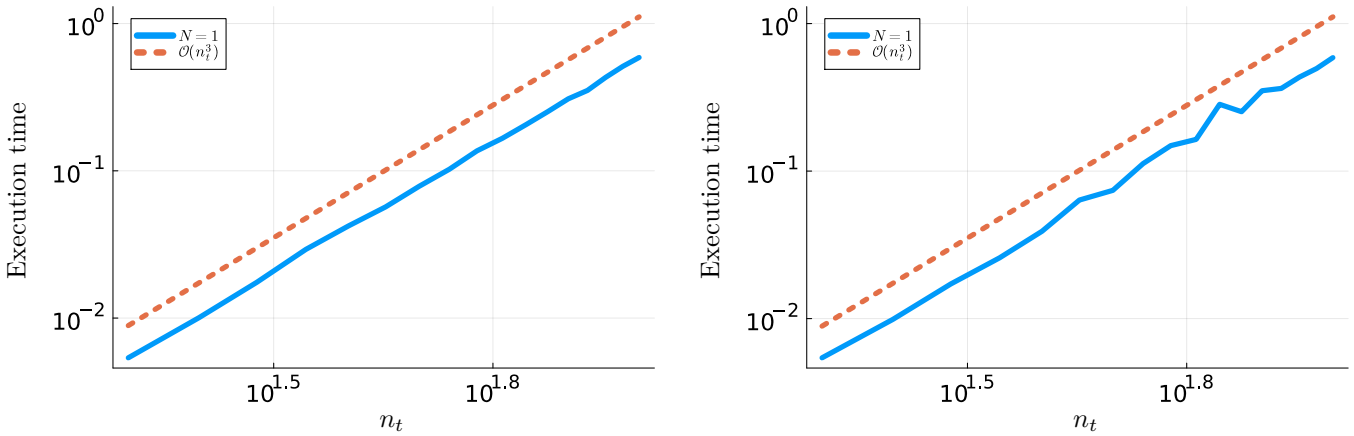
(b) Pointwise absolute errors for $n_x = 20$

Figure 4.3: (a) The condition number of the matrix in (4.17) grows with n, limiting achievable precision. (b) The pointwise absolute errors at final time $t = 1$ confirms that varying N in the N-step scheme consistently achieves errors at $\mathcal{O}(10^{-14})$.

Figure 4.3(b) depicts pointwise absolute errors between the numerical solution and the exact solution at the final time $t = 1$ across the spatial domain $[-5, 1]$, discretized with $K = 601$ grid points. The temporal degrees per subproblem in the N-step approach were chosen as $n_t(N = 1) = 20$, $n_t(N = 2) = 10$, $n_t(N = 4) = 9$, $n_t(N = 5) = 8$, and $n_t(N = 10) = 8$, where $n_x = 20$ is fixed for all values of N. These parameter selections ensured errors remained near $\mathcal{O}(10^{-14})$. For $N \geq 2$, the N-step spectral time-stepping methods achieve highly accurate solutions while substantially reducing computational time relative to the global spectral-in-time method, as highlighted in Figure 4.4(a).

Figure 4.4(a) presents a log-log plot of the execution time for the global solve with $n_x = 20$. As expected, the global spectral-in-time method exhibits cubic computational complexity in n_t, making it computationally expensive. In contrast, the N-step method ($N \geq 2$) significantly reduces execution time while preserving comparable accuracy. The performance improvement highlights the efficiency advantages of employing the N-step spectral time-stepping approach for $N \geq 2$.

For our next problem without an explicit solution, we again consider the linearized KdV equation, this time



(a) Execution time (forced linearized KdV)

(b) Execution time (linearized KdV)

Figure 4.4: Log-log plots of the global solve as the temporal resolution n_t increases, with fixed $n_x = 20$. (a) Forced linearized KdV equation. (b) Linearized KdV equation without the forcing term.

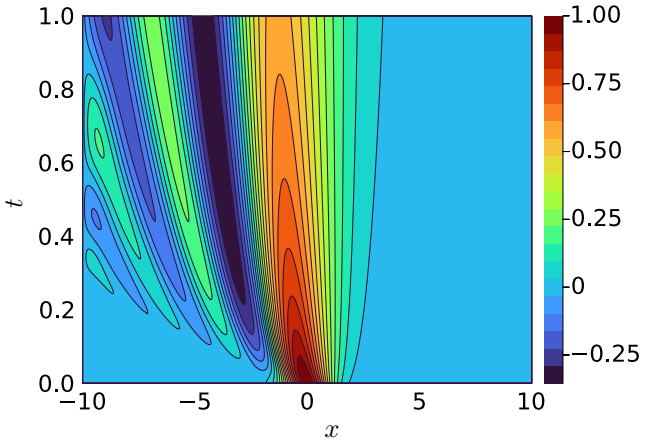
on a larger spatial domain and without a forcing term:

$$u_t + u_{xxx} = 0, \quad (x,t) \in [-10,10] \times [0,1], \tag{4.54}$$

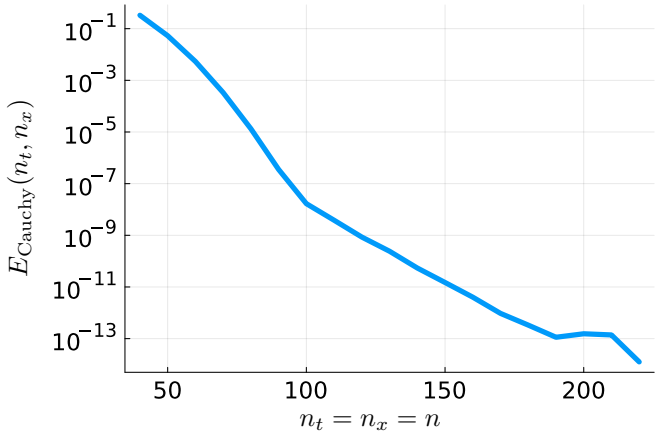
subject to boundary conditions $u(-10,t) = u(10,t) = u_x(10,t) = 0$ and the initial condition $u(x,0) = e^{-x^2}$. The `Julia` code used is as follows.

```
gd_t = UltraMappedInterval(t_0, t_0 + dt, 0.0); sp_t = Ultraspherical(0, gd_t)
sp1_t = Ultraspherical(1, gd_t)
gd_x = UltraMappedInterval(-10, 10, 0.0); sp_x = Ultraspherical(0, gd_x)
sp3_x = Ultraspherical(3, gd_x)
A1 = Matrix(Derivative() * sp_t, n_t-1, n_t); A2 = Matrix(Conversion(sp1_t)*sp_t, n_t-1, n_t)
C1 = Matrix(Conversion(sp3_x)*sp_x, n_x-3, n_x); C2 = Matrix(Derivative(3)*sp_x, n_x-3, n_x)
B_t = reshape([(-1)^(j+1) * sqrt(2) for j in 1:n_t],1,n_t); B_t[1] = 1
B_x1 = Matrix(Conversion(FixedGridValues([-10,10], gd_x)) * sp_x, 2, n_x)
B_x2 = Matrix(Conversion(FixedGridValues([10],gd_x))*Derivative()*sp_x,1,n_x)
Z1 = spzeros(1, n_t); Z2 = spzeros(3, n_x); Z3 = spzeros(n_t-1, n_t); Z4 = spzeros(n_x-3, n_x)
idx = sparse(I, n_x, n_x); idt = sparse(I, n_t, n_t)
AA1 = vcat(Z1,A1); AA2 = vcat(Z1,A2); CC1 = vcat(Z2,C1); CC2 = vcat(Z2,C2)
BBt = vcat(B_t, Z3); BBx = vcat(B_x1,B_x2, Z4)
L = kron(idx, BBt) + kron(BBx, idt) + kron(CC1, AA1) + kron(CC2, AA2)
F = zeros(n_t,n_x); F[1,:] = transpose(u0); bl, bu = bandwidths(kron(CC1,AA1) + kron(CC2,AA2))
k = 3*n_t; B,Ub,V = generate_data2(L, n_t*n_x, vec(F), k, bu)
```

```
XX = woodbury_sparseqr(B, Ub, V, k, bl); X = reshape(XX, n_t, n_x)
```



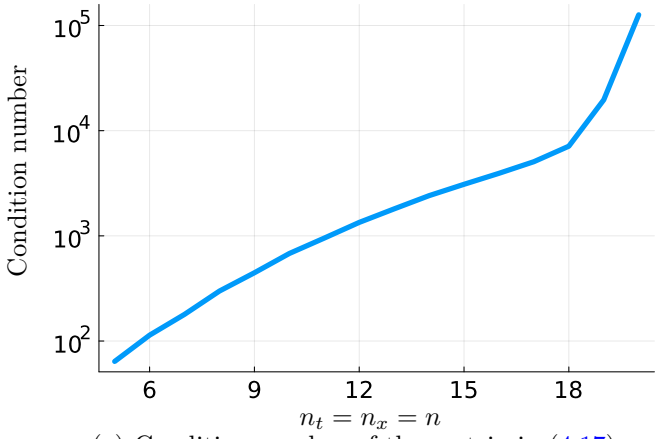
(a) Contour plot of the numerical solution



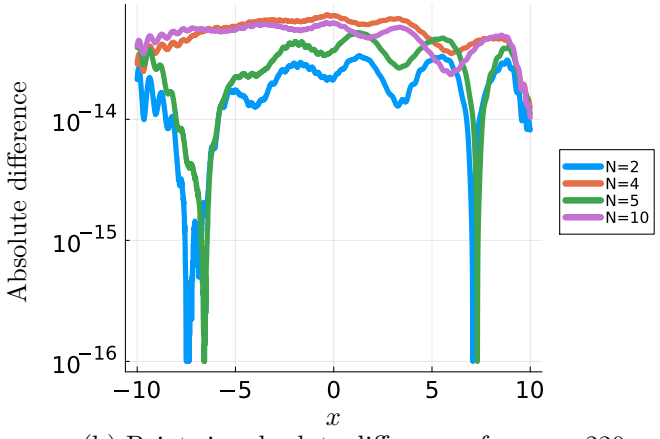
(b) Coefficient Cauchy errors

Figure 4.5: (a) Contour plot of the numerical solution on $(x, t) \in [-10, 10] \times [0, 1]$ for $n_t = n_x = 200$. (b) The coefficient Cauchy error $E_{\text{Cauchy}}(n_t, n_x)$, defined as in (4.49), decays to $\mathcal{O}(10^{-14})$ when $n \approx 220$.

Figure 4.5(a) shows the contour plot of the numerical solution obtained for $n_x = n_t = 200$. The solution shows a clear dispersion due to the term u_{xxx}. Figure 4.5(b) shows the Cauchy error $E_{\text{Cauchy}}(n_t, n_x)$ for the global solve, which decreases to $\mathcal{O}(10^{-14})$ at $n \approx 220$. Like the previous problem, the condition number is also relatively large, as shown in Figure 4.6(a).



(a) Condition number of the matrix in (4.17)



(b) Pointwise absolute differences for $n_x = 220$

Figure 4.6: (a) Condition number of the discretization matrix from (4.17) as n increases. (b) The pointwise absolute differences remain at $\mathcal{O}(10^{-14})$ for different values of N.

In Figure 4.6(b), we show the reference-based pointwise absolute errors of the numerical solutions over the spatial domain $[-10, 10]$, discretized with $K = 2001$ grid points. The temporal degrees per subproblem in the

N-step approach were chosen as $n_t(N=1)=220$, $n_t(N=2)=110$, $n_t(N=4)=55$, $n_t(N=5)=44$, and $n_t(N=10)=22$, where $n_x=220$ is fixed for all values of N. These results indicate that the N-step spectral time-stepping method consistently achieves errors of $\mathcal{O}(10^{-14})$, significantly reducing computational complexity compared to the global spectral-in-time approach. Figure 4.4(b) illustrates this improvement explicitly, highlighting the cubic computational complexity of the global spectral-in-time approach with respect to temporal resolution n_t, while demonstrating substantial computational savings using the N-step ($N \geq 2$) spectral time-stepping approach, which achieves similar accuracy with considerably reduced execution time. Such an efficiency is particularly beneficial for problems that require high-degree spatial discretizations (n_x is large), such as those explored in Section 4.4.3. Nevertheless, in the following section, we will see that selecting basis functions to yield purely banded matrices allows a single global solve to become computationally feasible.

4.4.2 Finite spatial domain with periodic boundary conditions

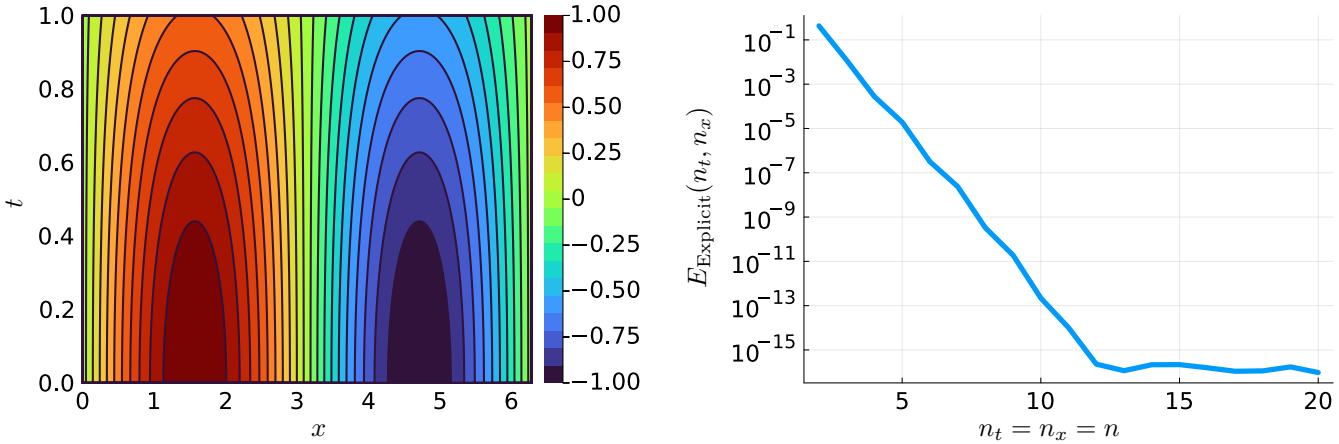
For our first problem with an explicit solution, we consider the following advection-diffusion equation with a forcing term:

$$u_t + u_{xxx} = -\sin(t)\sin(x) - \cos(t)\cos(x), \quad (x,t) \in [0,2\pi] \times [0,1], \tag{4.55}$$

subject to periodic boundary conditions $u(0,t) = u(2\pi,t), u_x(0,t) = u_x(2\pi,t), u_{xx}(0,t) = u_{xx}(2\pi,t)$ and the initial condition $u(x,0) = \sin(x)$. The explicit solution is given by $u(x,t) = \cos(t)\sin(x)$. The `Julia` code used is as follows.

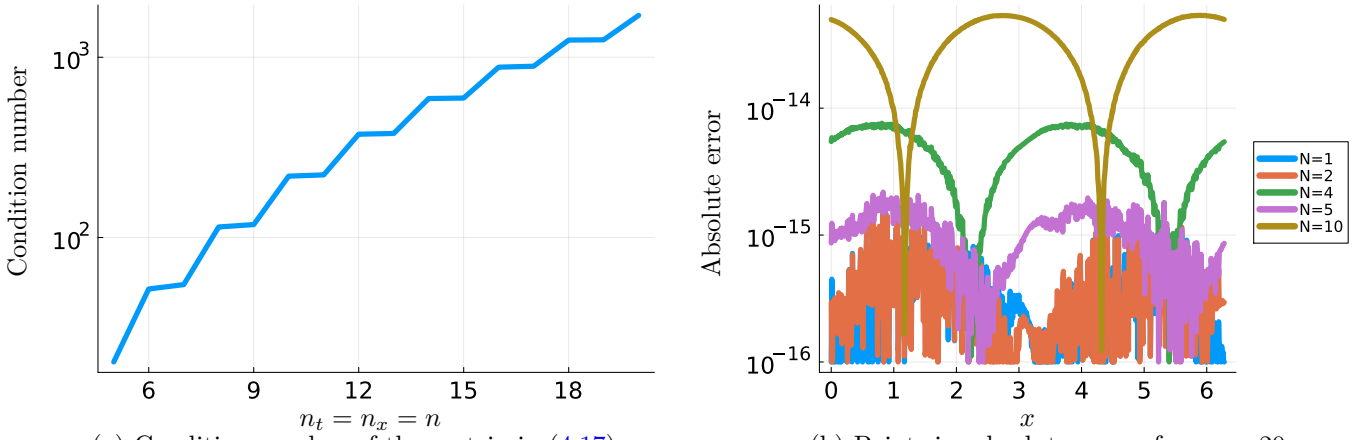
```
gd = UltraMappedInterval(t_0,t_0+Δt,0.0); sp = Ultraspherical(0, gd); sp1 = Ultraspherical(1, gd)
gdf = PeriodicMappedInterval(0,π2); spf = Fourier(gdf)
C1 = sparse(I,n_x,n_x); C2 = Matrix(Derivative(3)*spf,n_x,n_x)
A1 = Matrix(Derivative()*sp,n_t-1,n_t); A2 = Matrix(Conversion(sp1)*sp,n_t-1,n_t)
B = reshape([(-1)^(j+1) * sqrt(2) for j in 1:n_t],1,n_t); B[1] = 1
id = sparse(I,n_x,n_x); Z1 = spzeros(1,n_t); Z2 = spzeros(n_t-1,n_t)
AA1 = vcat(Z1,A1); AA2 = vcat(Z1,A2); BB = vcat(B,Z2); F = zeros(ComplexF64, n_t,n_x)
c_n = BasisExpansion(t -> -sin(t), sp1, n_t-1).c; c_k = BasisExpansion(x -> sin(x), spf, n_x).c
c_n2 = BasisExpansion(t -> -cos(t), sp1, n_t-1).c; c_k2 = BasisExpansion(x -> cos(x), spf, n_x).c
F_2 = c_n2 * transpose(c_k2) + c_n * transpose(c_k); F[2:end,1:end] = F_2; F[1,:] = transpose(u0)
L = kron(id,BB) + kron(C1,AA2) + kron(C2,AA2) + kron(id,AA1)
bl, bu = bandwidths(L); XX = sparse_qr(L, vec(F), bl); X = reshape(XX, n_t, n_x)
```

In Figure 4.7(a), the contour plot of the exact solution $u(x,t)$ is shown, illustrating spatial oscillations with a period of 2π. Figure 4.7(b) shows the coefficient error $E_{\text{Explicit}}(n_t, n_x)$ for the global solve, which decreases exponentially all the way to machine precision for polynomial degrees up to $n \approx 12$, confirming spectral convergence due to the smoothness of the solution. This behavior can also be demonstrated in Figure 4.8(a), which shows that the discretization matrix used is relatively well-conditioned.



(a) Contour plot of $u(x,t)$ (b) Coefficient errors for $\tilde{n}_t = \tilde{n}_x = 20$

Figure 4.7: (a) Contour plot of the exact solution $u(x,t) = \cos(t)\sin(x)$ on $(x,t) \in [0, 2\pi] \times [0, 1]$. (b) The coefficient error $E_{\text{Explicit}}(n_t, n_x)$, defined as in (4.46), decays to machine precision at $n \approx 12$.

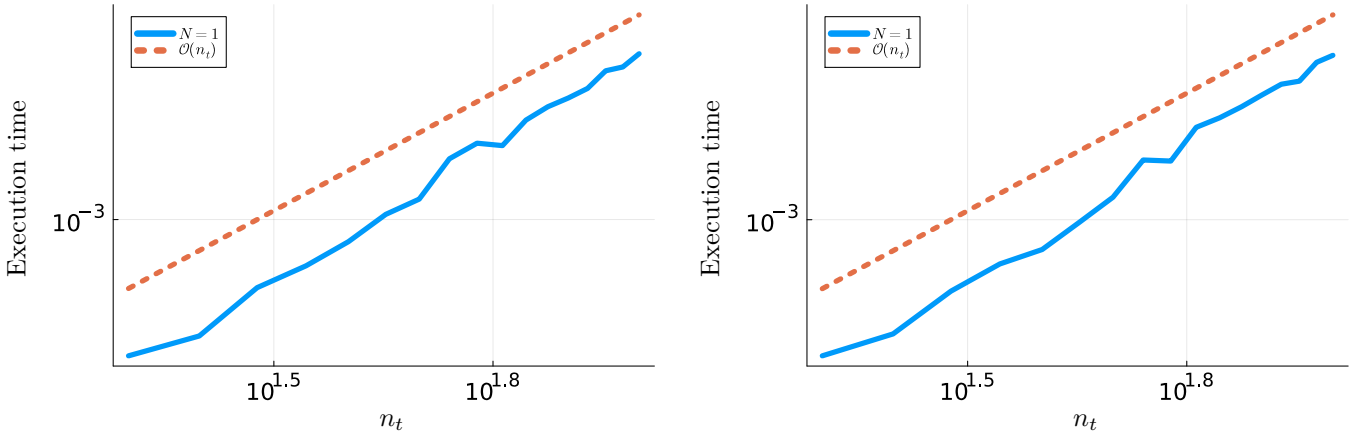


(a) Condition number of the matrix in (4.17) (b) Pointwise absolute errors for $n_x = 20$

Figure 4.8: (a) Condition number of the discretization matrix from (4.17) as n increases. (b) The pointwise absolute errors remain at $\mathcal{O}(10^{-14})$ for different values of N.

In Figure 4.8(b), we plot the pointwise absolute errors between the numerical solutions and the exact solution over the spatial domain $[0, 2\pi]$ at $t = 1$, discretized using $K = 629$ grid points. The temporal degrees per subproblem in the N-step approach were chosen as $n_t(N = 1) = 20$, $n_t(N = 2) = 10$, $n_t(N = 4) = 8$, $n_t(N = 5) = 8$, and $n_t(N = 10) = 7$, ensuring that the errors at the final time $t = 1$ remain at $\mathcal{O}(10^{-14})$.

Figure 4.9(a) shows that the execution time of the global solve increases linearly as n_t increases. This outcome arises because the discretization matrix for this particular problem is purely banded, and thus the computational complexity of solving the associated linear systems via Algorithm 5 scales linearly with n_t.



(a) Execution time (forced linearized KdV equation)

(b) Execution time (linearized KdV equation)

Figure 4.9: Log-log plots of the global solve as the temporal resolution n_t increases, with fixed $n_x = 20$. (a) Forced linearized KdV equation. (b) Linearized KdV equation without the forcing term.

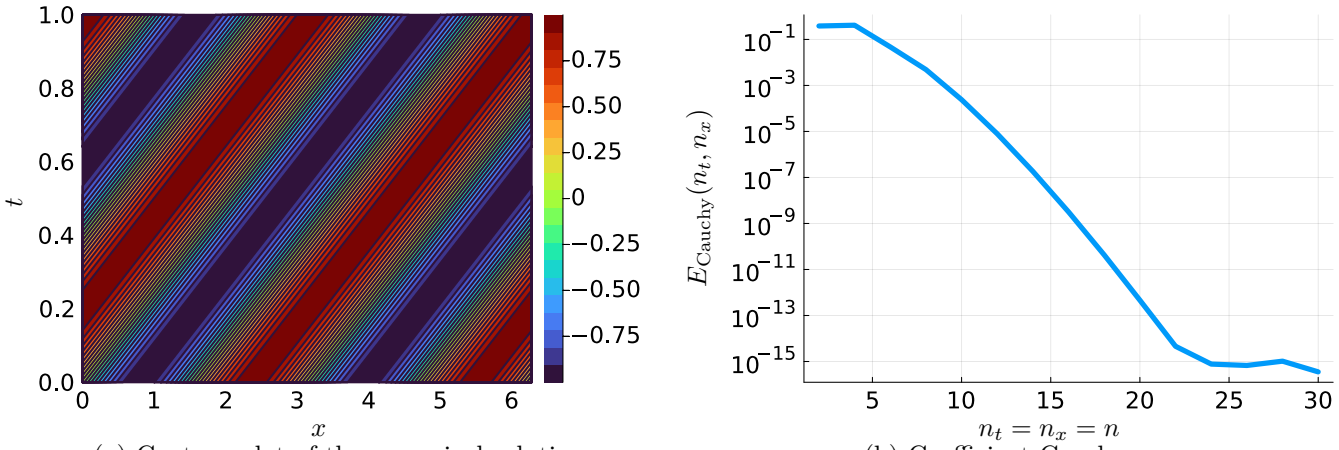
For our next problem without an explicit solution, we can still consider this problem but without the forcing term:

$$u_t + u_{xxx} = 0, \quad (x,t) \in [0, 2\pi] \times [0, 1], \tag{4.56}$$

subject to periodic boundary conditions $u(0,t) = u(2\pi,t), u_x(0,t) = u_x(2\pi,t), u_{xx}(0,t) = u_{xx}(2\pi,t)$ and the initial condition $u(x,0) = \sin(2x)$. The `Julia` code used is as follows, which is the same as the previous one except now without the forcing term on the right-hand side.

```
gd = UltraMappedInterval(t_0,t_0+Δt,0.0); sp = Ultraspherical(0, gd); sp1 = Ultraspherical(1, gd)
gdf = PeriodicMappedInterval(0,π2); spf = Fourier(gdf)
C1 = sparse(I,n_x,n_x); C2 = Matrix(Derivative(3)*spf,n_x,n_x)
A1 = Matrix(Derivative()*sp,n_t-1,n_t); A2 = Matrix(Conversion(sp1)*sp,n_t-1,n_t)
B = reshape([(-1)^(j+1) * sqrt(2) for j in 1:n_t],1,n_t); B[1] = 1
id = sparse(I,n_x,n_x); Z1 = spzeros(1,n_t); Z2 = spzeros(n_t-1,n_t)
AA1 = vcat(Z1,A1); AA2 = vcat(Z1,A2); BB = vcat(B,Z2)
F = zeros(n_t-1,n_x); F = vcat(transpose(u0),F); L = kron(id,BB) + kron(C2,AA2) + kron(C1,AA1)
bl, bu = bandwidths(L); XX = sparse_qr(L, vec(F), bl); X = reshape(XX, n_t, n_x)
```

Figure 4.10(a) shows the contour plot of the numerical solution computed with $n_t = n_x = 29$. Figure 4.10(b) shows the coefficient Cauchy error $E_{\text{Cauchy}}(n_t, n_x)$ for the global solve, which decreases all the way to machine precision for polynomial degrees up to $n \approx 22$. Figure 4.11(a) shows that the discretization matrix used is relatively well-conditioned since it is a purely banded matrix (it is the same matrix as the previous one, we plot it just for completeness).



(a) Contour plot of the numerical solution

(b) Coefficient Cauchy errors

Figure 4.10: (a) Contour plot of the numerical solution on $(x, t) \in [0, 2\pi] \times [0, 1]$ for $n_t = n_x = 29$. (b) The coefficient Cauchy error $E_{\text{Cauchy}}(n_t, n_x)$, defined as in (4.49), decays to machine precision at $n \approx 22$.

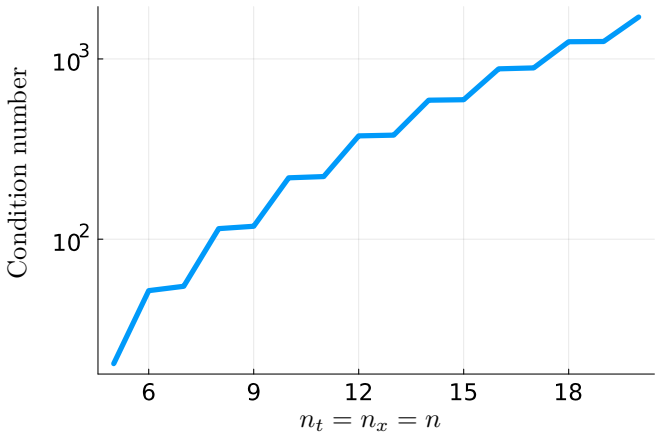
In Figure 4.11(b), we plot the pointwise absolute differences over the spatial domain $[0, 2\pi]$ at $t = 1$, discretized using $K = 629$ grid points. The temporal degrees per subproblem in the N-step approach were chosen as $n_t(N = 1) = 20$, $n_t(N = 2) = 10$, $n_t(N = 4) = 8$, $n_t(N = 5) = 8$, and $n_t(N = 10) = 7$, ensuring that the errors at the final time $t = 1$ remain consistently at $\mathcal{O}(10^{-14})$. Like the case with a forcing term, Figure 4.9(b) shows that the computational complexity of one global solve via Algorithm 5 scales linearly with n_t.

4.4.3 Unbounded spatial domain

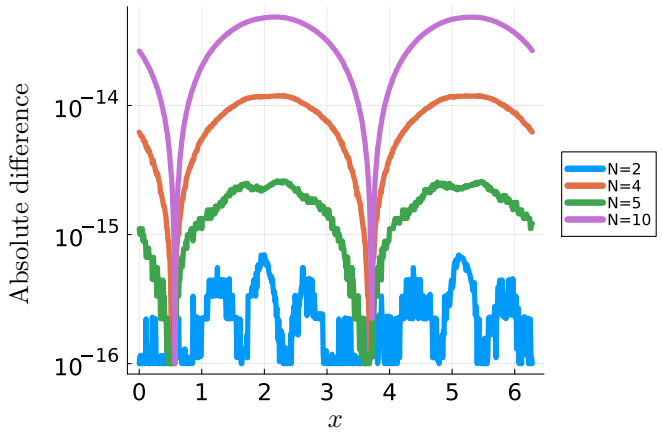
For our first explicit-solution example, we consider the following forced linearized KdV equation:

$$u_t + u_{xxx} = e^{-x^2}\left[(12x - 8x^3)\cos(t) - \sin(t)\right], \quad (x, t) \in \mathbb{R} \times [0, 1], \tag{4.57}$$

subject to boundary conditions $u(x, t) \to 0$ as $x \to \pm\infty$ and the initial condition $u(x, 0) = e^{-x^2}$. The exact solution is given by $u(x, t) = e^{-x^2}\cos(t)$. The `Julia` code used to solve this problem is as follows.



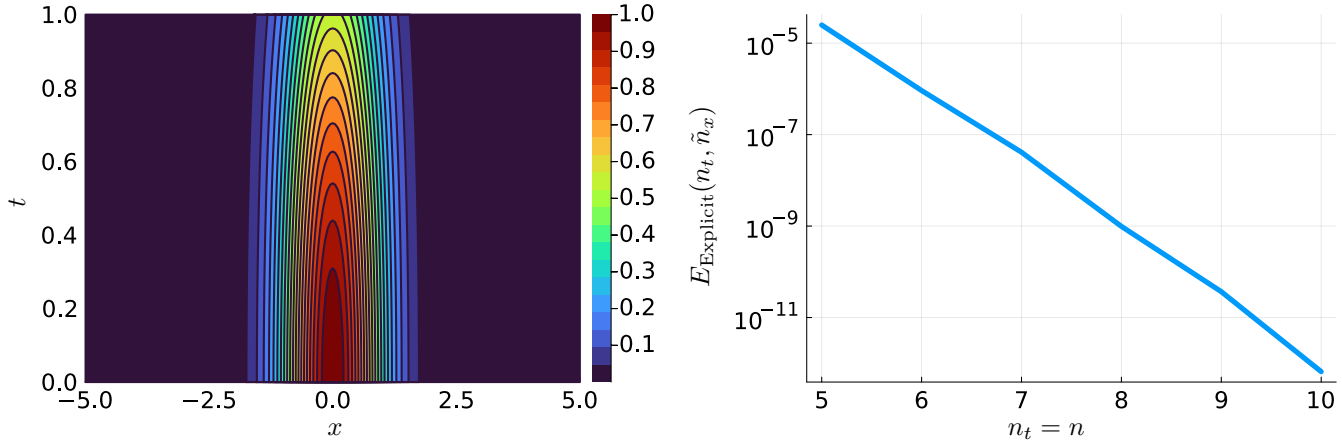
(a) Condition number of the matrix in (4.17)



(b) Pointwise absolute differences for $n_x = 20$

Figure 4.11: (a) Condition number of the discretization matrix from (4.17) as n increases. (b) The pointwise absolute differences remain at $\mathcal{O}(10^{-14})$ for different values of N.

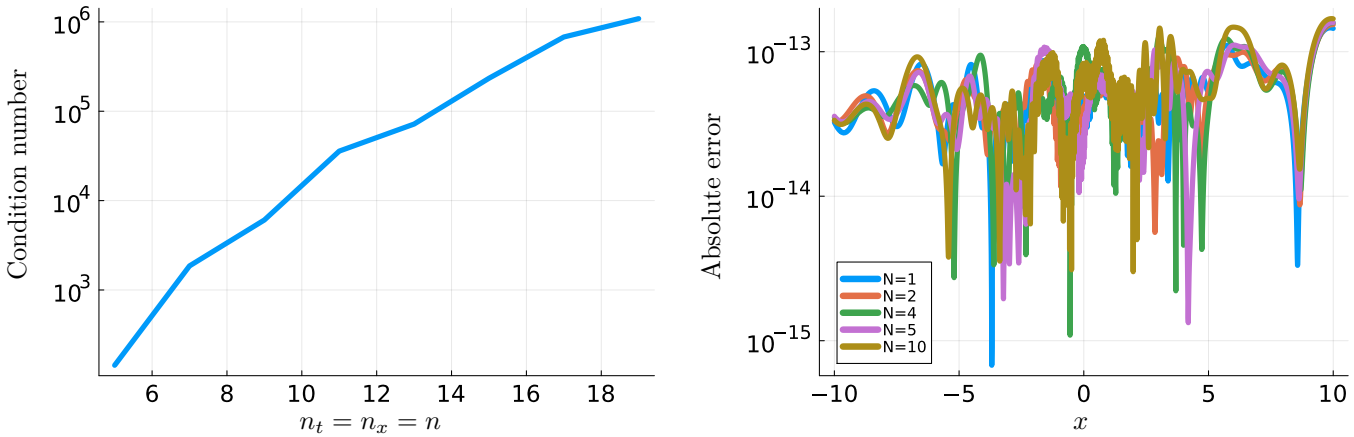
```
gdt = UltraMappedInterval(t_0,t_0+Δt,0.0)
spt = Ultraspherical(0, gdt); spt1 = Ultraspherical(1, gdt)
gdx = RationalRealAxis(); spx = OscRational(gdx,0.0)
C1 = sparse(I,n_x,n_x)[2:end,1:end]; C2 = Matrix(Derivative(3)*spx,n_x - 1,n_x)
Bx = ones(1,n_x)
A1 = Matrix(Derivative(1)*spt,n_t-1,n_t); A2 = Matrix(Conversion(spt1)*spt,n_t-1,n_t)
Bt = reshape([(-1)^(j+1) * sqrt(2) for j in 1:n_t],1,n_t); Bt[1] = 1
id = sparse(I,n_x,n_x); id2 = sparse(I,n_t,n_t)
Z1 = spzeros(1,n_t); Z2 = spzeros(n_t-1,n_t); Z3 = spzeros(1,n_x); Z4 = spzeros(n_x-1,n_x)
AA1 = vcat(Z1,A1); AA2 = vcat(Z1,A2)
CC1 = vcat(Z3,C1); CC2 = vcat(Z3,C2)
BBt = vcat(Bt,Z2); BBx = vcat(Bx,Z4)
F = zeros(ComplexF64,n_t,n_x)
c_n = BasisExpansion(t -> cos(t), spt1, n_t-1).c
c_k = BasisExpansion(x -> (12*x-8*x^3)*exp(-x^2), spx, n_x-1).c
c_n2 = BasisExpansion(t -> -sin(t), spt1, n_t-1).c
c_k2 = BasisExpansion(x -> exp(-x^2), spx, n_x-1).c
F_2 = c_n2 * transpose(c_k2) + c_n * transpose(c_k); F[2:end,2:end] = F_2; F[1,:] = transpose(u0)
L = kron(id,BBt) + kron(CC1,AA1) + kron(CC2,AA2) + kron(BBx,id2)
bl, bu = bandwidths(kron(CC1,AA1) + kron(CC2,AA2))
k = n_t; B,Ub,V = generate_data2(L,n_t*n_x,vec(F),k,bu)
XX = woodbury_sparseqr(B, Ub, V, k, bl); X = reshape(XX, n_t, n_x)
```



(a) Contour plot of $u(x,t)$ (b) The temporal coefficient error for $\tilde{n}_x = 200$

Figure 4.12: (a) Contour plot of the exact solution $u(x,t) = e^{-x^2}\cos(t)$ on $(x,t) \in [-5,5] \times [0,1]$. (b) The temporal coefficient error decays rapidly to $\mathcal{O}(10^{-12})$ at $n_t = 10$. The spatial resolution is fixed at $n_x = \tilde{n}_x = 200$.

Figure 4.12(a) shows the contour plot of the exact solution on $(x,t) \in [-5,5] \times [0,1]$. Figure 4.12(b) shows the decay of the temporal coefficients. Here, the coefficient error $E_{\text{Explicit}}(n_t, \tilde{n}_x)$ is shown as the degree of the temporal polynomial n_t increases, while the spatial resolution remains fixed at $n_x = 200$. The decision to fix the spatial resolution is deliberate: since the solution is defined over an infinite spatial domain and rapidly becomes highly localized around the origin, a sufficiently large number of rational basis functions are essential to accurately resolve these steep spatial gradients. Therefore, the coefficient of every additional spatial degree is relatively large, which would introduce substantial errors and severely compromise the accuracy.



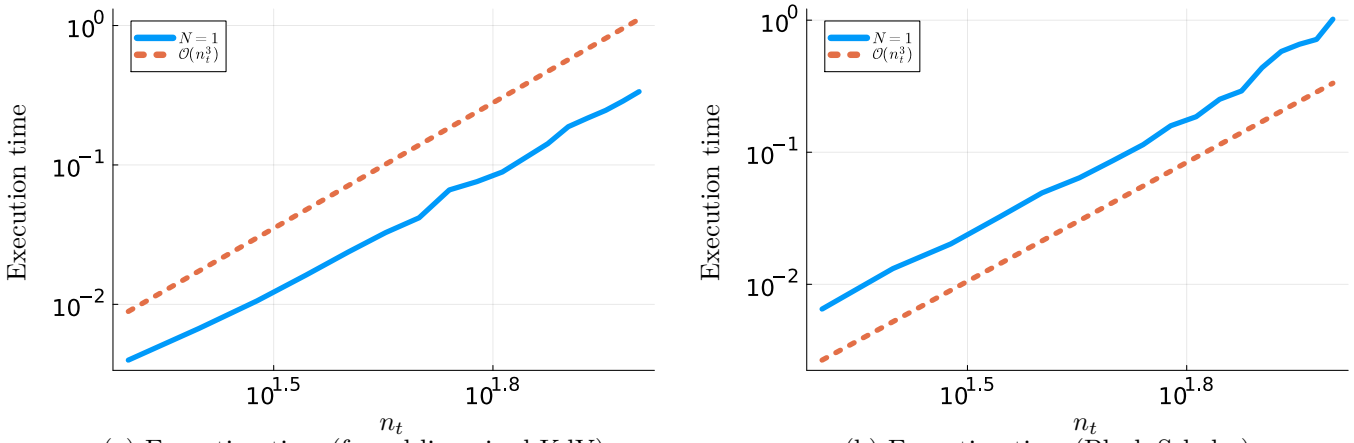
(a) Condition number of the matrix in (4.17) (b) Pointwise absolute errors for $n_x = 200$

Figure 4.13: (a) Condition number of the discretization matrix from (4.17) as n increases. (b) Pointwise absolute errors at $t = 1$ computed with different N-step spectral time-stepping methods, maintaining consistently high accuracy at $\mathcal{O}(10^{-13})$ across the domain $[-10, 10]$.

Figure 4.13(a) shows the growth of the condition number for the discretization matrix as n increases. Fig-

ure 4.13(b) shows the pointwise absolute errors at the final time $t = 1$ between the numerical solutions and the exact solution, computed over the spatial interval $[-10, 10]$ discretized into $K = 2001$ points. The temporal degrees per subproblem in the N-step approach were chosen as $n_t(N = 1) = 20$, $n_t(N = 2) = 12$, $n_t(N = 4) = 8$, $n_t(N = 5) = 8$, and $n_t(N = 10) = 7$. These parameters are carefully selected to ensure uniformly high accuracy across the domain, with errors consistently reaching impressive $\mathcal{O}(10^{-13})$.

Figure 4.14(a) shows the execution time of the global solve, revealing a cubic complexity scaling with respect to n_t. This efficiency gain is especially pronounced for problems requiring high spatial resolution (e.g., $n_x = 200$), where using a global solve would be computationally prohibitive.



(a) Execution time (forced linearized KdV)

(b) Execution time (Black-Scholes)

Figure 4.14: Log-log plots of the global solve as the temporal resolution n_t increases, with fixed $n_x = 20$. (a) Forced linearized KdV equation. (b) Black-Scholes equation.

For our next problem without an explicit solution, we consider the classical Black-Scholes equation:

$$u_t + x^2 u_{xx} + x u_x - u = 0, \quad (x, t) \in \mathbb{R} \times [0, 1], \tag{4.58}$$

subject to boundary conditions $u(x, t) \to 0$ as $x \to \pm\infty$ and the initial condition $u(x, 1) = x^{10} e^{-10x^2}$. Although the exact solution can be formally represented as:

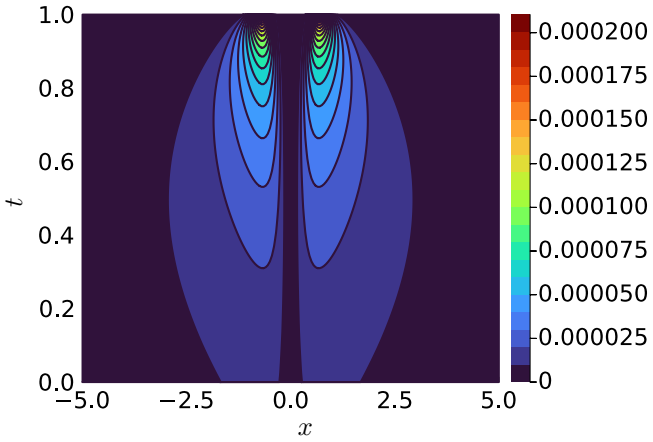
$$u(x, t) = \frac{e^{t-1}}{\sqrt{4\pi(1-t)}} \int_{-\infty}^{\infty} \exp\left(-\frac{(\ln|x| - \xi)^2}{4(1-t)}\right) \exp\left(10\xi - 10e^{2\xi}\right) d\xi, \tag{4.59}$$

it is computationally challenging to obtain its bivariate expansion coefficients with high precision. Thus, for practical purposes, we treat this problem as one without an explicit solution. The `Julia` code is as follows.

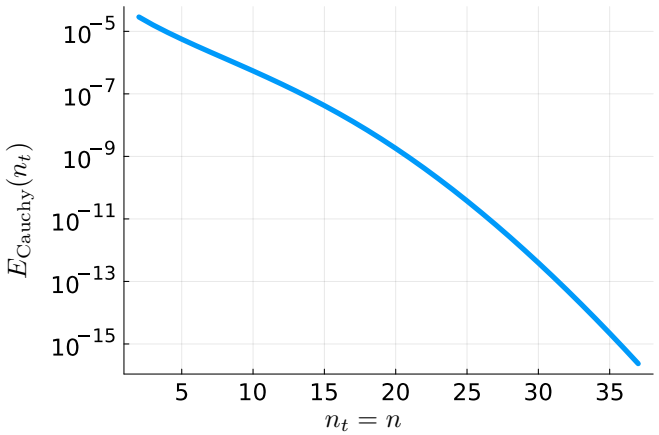
```
gdt = UltraMappedInterval(t_0,t_0 + dt,0.0)
spt = Ultraspherical(0, gdt); spt1 = Ultraspherical(1, gdt)
gdx = RationalRealAxis(); spx = OscRational(gdx,0.0)
xw = OperatorApproximation.pad(BasisExpansion(x -> 2im*x/(x + 1im)^2, spx), 7)
Mx = Multiplication(xw)*spx
wD = OperatorApproximation.WeightedDerivative(spx)
C1 = sparse(I,n_x,n_x)[2:end,1:end]; C = Matrix(Mx*wD,n_x,n_x); C22 = C^2 - C
C2 = C22[2:end,1:end] + Matrix(Mx*wD, n_x - 1, n_x) - C1
Bx = ones(1,n_x)
A1 = Matrix(Derivative(1)*spt,n_t-1,n_t); A2 = Matrix(Conversion(spt1)*spt,n_t-1,n_t)
Bt = ones(1,n_t)*√2; Bt[1] = 1
id = sparse(I,n_x,n_x); id2 = sparse(I,n_t,n_t)
Z1 = spzeros(1,n_t); Z2 = spzeros(n_t-1,n_t); Z3 = spzeros(1,n_x); Z4 = spzeros(n_x-1,n_x)
AA1 = vcat(Z1,A1); AA2 = vcat(Z1,A2)
CC1 = vcat(Z3,C1); CC2 = vcat(Z3,C2)
BBt = vcat(Bt,Z2); BBx = vcat(Bx,Z4)
F = zeros(ComplexF64,n_t-1,n_x); F = vcat(transpose(u0),F)
L = kron(id,BBt) + kron(CC1,AA1) + kron(CC2,AA2) + kron(BBx,id2)
bl, bu = bandwidths(kron(CC1,AA1) + kron(CC2,AA2))
k = n_t; B,Ub,V = generate_data2(L,n_t*n_x,vec(F),k,bu)
XX = woodbury_sparseqr(B, Ub, V, k, bl); X = reshape(XX, n_t, n_x)
```

Figure 4.15(a) shows the contour plot of the numerical solution on $(x,t) \in [-5,5] \times [0,1]$ for $n_t = 30$ and $n_x = 301$. Because the PDE and initial condition are even in x, the solution remains symmetric about $x = 0$. The solution also remains highly localized in the x-direction, rapidly decaying to machine precision near $|x| \approx 5$. Figure 4.15(b) shows the decay of the temporal coefficients. Here, the Cauchy error $E_{\text{Cauchy}}(n_t)$ is shown as the degree of the temporal polynomial n_t increases, while the spatial resolution remains fixed at $n_x = 130$. The plot shows rapid spectral convergence in the temporal dimension, with error decreasing to machine precision at $n_t \approx 35$.

More details on conditioning and pointwise accuracy are shown in Figure 4.16. In Figure 4.16(a), we track the condition number of the discretization matrix for various n. The condition number remains at or below $\mathcal{O}(10^2)$ up to $n \approx 20$, suggesting that the system does not become severely ill-conditioned at these moderate degrees. Figure 4.16(b) shows the pointwise absolute differences at the final time $t = 0$, computed over the spatial interval



(a) Contour plot of the numerical solution

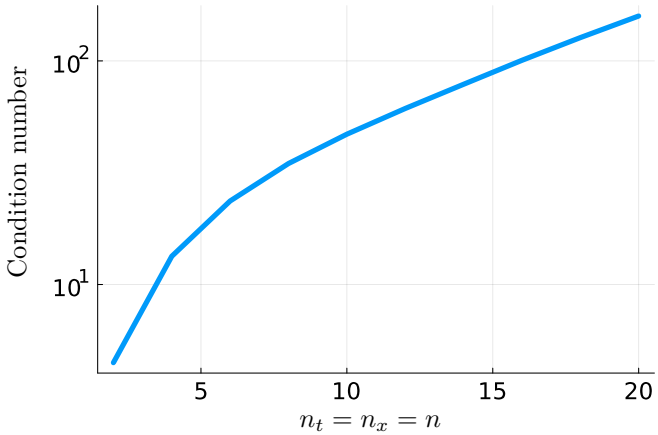


(b) The temporal coefficient Cauchy error for $n_x = 130$

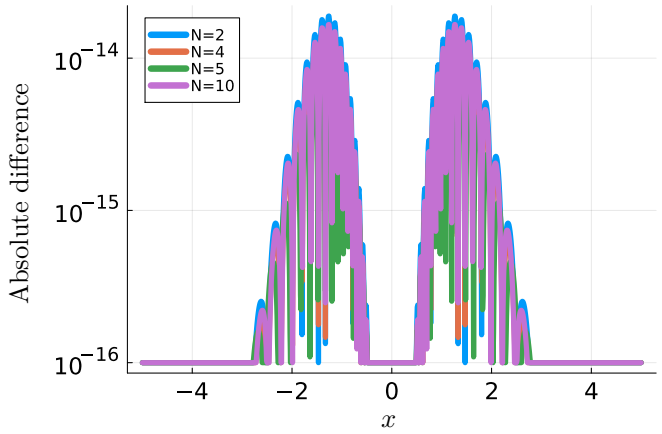
Figure 4.15: (a) Contour plot of the numerical solution on $(x, t) \in [-5, 5] \times [0, 1]$ for $n_t = 30$ and $n_x = 301$. (b) The temporal coefficient Cauchy error $E_{\text{Cauchy}}(n_t)$, defined as in (4.50), decays rapidly to machine precision at $n_t \approx 35$. The spatial resolution is fixed at $n_x = 130$.

$[-5, 5]$ discretized into $K = 1001$ points. The temporal degrees per subproblem in the N-step approach were chosen as $n_t(N = 1) = 90$, $n_t(N = 2) = 45$, $n_t(N = 4) = 23$, $n_t(N = 5) = 18$, and $n_t(N = 10) = 9$. The resulting errors remain at $\mathcal{O}(10^{-15})$.

Finally, Figure 4.14(b) shows the computational complexity of the global solve, revealing a cubic complexity scaling with respect to n_t.



(a) Condition number of the matrix in (4.17)



(b) Pointwise absolute differences for $n_x = 130$

Figure 4.16: (a) Condition number of the discretization matrix from (4.17) as n increases. (b) Pointwise absolute differences at $t = 1$ computed with different N-step spectral time-stepping methods, maintaining consistently high accuracy at $\mathcal{O}(10^{-14})$ across the domain $[-5, 5]$.

Chapter 5

Computing the Tracy-Widom distribution for arbitrary $\beta > 0$

One application of the spectral time-stepping method is to compute the numerical solution of the Bloemendal–Virág equation to machine precision, which is only feasible through time-stepping, as a global-in-time solve would either fail to meet the desired precision or impose huge storage requirements due to the relatively large number of basis functions needed in the spatial domain. Before we go into the numerical approach, we first introduce the Bloemendal–Virág equation and motivate its study.

5.1 Introduction

The solution of the Bloemendal-Virág equation is closely related to the famous Tracy-Widom distribution, named after Tracy and Widom [TW93, TW94, TW96], which gives the limiting distribution of the rescaled largest eigenvalue of a random matrix taken from an appropriate symmetry class. More precisely, the largest eigenvalue $\lambda_{\max}$ satisfies the following fundamental limit

$$\lim_{n\to\infty} \mathbb{P}\left(n^{1/6}\left(\lambda_{\max}\left(A_n\right) - 2\sqrt{n}\right) \le x\right) = F_\beta\left(x\right) \text{ for all } x \in \mathbb{R}, \tag{5.1}$$

where F_β is the Tracy-Widom distribution and $\beta = 1,\ 2,\ 4$ if $A_n \sim$ Gaussian Orthogonal Ensemble, Gaussian Unitary Ensemble, Gaussian Symplectic Ensemble, respectively.

For an $n \times n$ Gaussian ensemble A_n with ordered eigenvalues $\{\lambda_j\}_{j=1}^n$, the joint probability density function (jpdf) for its eigenvalues is given by

$$\rho\left(\lambda_1, \lambda_2, \cdots, \lambda_n\right) = Z_{n,\beta}^{-1} \prod_i e^{-\beta\lambda_i^2/4} \prod_{i<j} \left|\lambda_j - \lambda_i\right|^\beta,\ \lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_n, \tag{5.2}$$

where $Z_{n,\beta}$ is the partition function. For $\beta = 1, 2, 4$, (5.2) is solvable: all correlation functions in finite dimensions can be explicitly calculated using Hermite polynomials. This provides definitive local limit theorems and establishes clear limits for the random points [Meh04]. Importantly, (5.2) also describes a one-dimensional Coulomb gas

at inverse temperature β for any $\beta > 0$. If $\beta \neq 1, 2, 4, 6$, there are no known explicit formulae which appear amenable to asymptotic analysis (see, for example, [Li18, GIKM16, Rum16] for $\beta = 6$). One, in general, must resort to numerical computations, see [Bor09] for $\beta = 1, 2, 4$.

Dumitriu and Edelman [DE02] established that a family of symmetric tridiagonal (Jacobi) matrix models have (5.2) as their eigenvalue density for any $\beta > 0$. More specifically, the matrix given by

$$H_n^\beta = \frac{1}{\sqrt{\beta}} \begin{pmatrix} g & \chi_{(n-1)\beta} & & & \\ \chi_{(n-1)\beta} & g & \chi_{(n-2)\beta} & \ddots & \\ & \chi_{(n-2)\beta} & g & \ddots & \chi_\beta \\ & & \ddots & \chi_\beta & g \end{pmatrix}, \tag{5.3}$$

has (5.2) as the jpdf of its eigenvalues. Here the entries are independent random variables, up to symmetry, $g \sim N(0, 2)$ and $\chi_k \sim \text{Chi}(k)$. We call H_n^β the β-Hermite ensemble.

Sutton and Edelman [Sut06, ES07] then presented an argument[1] describing how the spectrum of the rescaled operator $\tilde{H}_n^\beta = n^{1/6}\left(2\sqrt{n}I - H_n^\beta\right)$, where I is the $n \times n$ identity matrix, should be described by the spectrum of the stochastic Airy operator $\mathcal{H}_\beta = -\frac{d^2}{dx^2} + x + \frac{2}{\sqrt{\beta}} b_x'$, as $n \to \infty$, where b_x' is standard Gaussian white noise.

As a result, the following eigenvalue problem was considered in [RRV11]

$$\mathcal{H}_\beta f = \Lambda f \text{ on } L^2(\mathbb{R}_+) \tag{5.4}$$

with a Dirichlet boundary condition $f(0) = 0$. Ramírez, Rider, and Virág [RRV11] proved the following theorem.

Theorem 5.1.1 (Theorem 1.1 in [RRV11]). *With probability one, for each $k \geq 0$, the set of eigenvalues of $\mathcal{H}_\beta$ has a well-defined $(k+1)$st lowest element Λ_k. Moreover, let $\lambda_1 \geq \lambda_2 \geq \cdots$ denote the eigenvalues of H_n^β. Then the vector $\left(n^{1/6}(2\sqrt{n} - \lambda_l)\right)_{l=1,\ldots,k}$ converges in distribution to $(\Lambda_0, \Lambda_1, \cdots, \Lambda_{k-1})$ as $n \to \infty$.*

Ramírez, Rider, and Virág showed that the distribution of $-\Lambda_0$ is a consistent definition of Tracy-Widom($\cdot$) for general $\beta > 0$. Based on [VV09, RRV11], Bloemendal and Virag [Blo11, BV13] considered the generalized eigenvalue problem (5.4) with boundary condition $f'(0) = \omega f(0)$, where $\omega \in \mathbb{R}$ represents a scaling parameter. Let $\mathcal{H}_{\beta,\omega}$ denote $\mathcal{H}_\beta$ together with this boundary condition. According to [RRV11], the distribution $F_{\beta,\omega}$ of $-\Lambda_0$ in the case $\omega = +\infty$ thus coincides with Tracy-Widom for general $\beta > 0$.

[1] A version of this argument was first presented by Edelman at the SIAM Conference on Applied Linear Algebra held at the College of William & Mary in 2003.

Bloemendal and Virág [BV13] further showed that $F = F_{\beta,\omega}$ can be characterized using the solution of a boundary value problem (BVP), i.e., the Bloemendal-Virág equation, as we now discuss.

The Tracy-Widom distribution function F_β can be characterized as follows [Blo11]. Consider

$$\frac{\partial F}{\partial x} + \frac{2}{\beta}\frac{\partial^2 F}{\partial \omega^2} + \left(x - \omega^2\right)\frac{\partial F}{\partial \omega} = 0, \quad (\omega, x) \in \mathbb{R}^2, \tag{5.5}$$

with boundary conditions given by

$$F(\omega, x) \to 1 \quad \text{as } x, \omega \to \infty \text{ together,} \tag{5.6}$$

$$F(\omega, x) \to 0 \quad \text{as } \omega \to -\infty \text{ with } x \text{ bounded above.} \tag{5.7}$$

Then [BV13, Theorem 1.7]

$$F_\beta(x) = \lim_{\omega \to \infty} F(x, \omega). \tag{5.8}$$

Using $\omega = -\cot\theta$, we rewrite (5.5) as

$$\frac{\partial H}{\partial x} + \left(\frac{2}{\beta}\sin^4\theta\right)\frac{\partial^2 H}{\partial \theta^2} + \left[\left(x + \frac{2}{\beta}\sin 2\theta\right)\sin^2\theta - \cos^2\theta\right]\frac{\partial H}{\partial \theta} = 0, \tag{5.9}$$

with boundary condition $H(x, 0) = 0$.

To address the boundary condition as $x \to \infty$ and $\theta \to \pi$, we truncate the domain at a finite value, denoted as $x = x_0 > 0$. We then employ the Gaussian asymptotics established by Bloemendal [Blo11, Theorem 4.1.1] to obtain the following approximate asymptotic initial condition [Blo11, p.103]:

$$H(\theta, x_0) = \begin{cases} \Phi\left(\frac{x_0 - \cot^2\theta}{\sqrt{(4/\beta)\cot\theta}}\right) & 0 \le \theta \le \pi/2, \\ 1 & \pi/2 \le \theta. \end{cases} \tag{5.10}$$

Here Φ denotes the standard normal distribution function. Note that $H(\theta, x_0)$ is continuous in both $x_0 > 0$ and $\theta \ge 0$. One then approximates the undeformed Tracy-Widom distribution $F_\beta(x)$ at $\theta = \pi$ by $\omega = -\cot\theta$ and (5.8), i.e., $F_\beta(x) \approx H(x, \pi)$.

5.2 Computation of the Tracy-Widom distribution using FD method

One way to solve this BVP is by discretizing it using finite differences; see, for example, [LeV07]. For $0 \le n \le N, 0 \le m \le M$, define

$$x_n = x_0 + n\Delta x, \quad \theta_m = mh. \tag{5.11}$$

Given that $\beta > 0$, one integrates equation (5.9) backward in "time" with respect to the time-like variable x to guarantee its well-posedness: $\Delta x < 0$. Denote the approximation of $H(\theta_m, x)$ by $H_m(x)$. We then replace the partial derivatives of H with respect to θ by centered differences, with grid spacing h. The method of lines formulation reads

$$\frac{\partial \mathbf{H}_M}{\partial x} = -\left(\frac{2\sin^4\theta}{\beta h^2}\right)\check{T}\mathbf{H}_M + \frac{\left[\left(x + \frac{2}{\beta}\sin 2\theta\right)\sin^2\theta - \cos^2\theta\right]}{2h}\check{U}\mathbf{H}_M, \tag{5.12}$$

where $\mathbf{H}_M(x) = [H_1(x), H_2(x), \cdots, H_M(x)]^T$, $Mh = \theta_M$, and

$$\check{T} = \begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{pmatrix}, \quad \check{U} = \begin{pmatrix} 0 & -1 & & & & \\ 1 & 0 & -1 & & & \\ & 1 & 0 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & -1 \\ & & & -1 & 4 & -3 \end{pmatrix}. \tag{5.13}$$

$\mathbf{H}_M$ excludes H_0 since $H_0(x) = 0$. The coefficients in the last row of $\check{U}$ come from the parameters of BDF2. Note that there is no need to replace the last row of $\check{T}$ using a BDF since $\theta_M = \pi$ and, mathematically, there is no need to set an additional boundary condition since (5.12) has a vanishing diffusivity for $\theta = \pi$.

Let

$$T(\beta, \theta_M, h) := -\left(\frac{2\sin^4\theta_M}{\beta h^2}\right)\check{T} = \begin{pmatrix} \frac{-2\sin^4\theta_1}{\beta h^2} & & \\ & \ddots & \\ & & \frac{-2\sin^4\theta_M}{\beta h^2} \end{pmatrix}\check{T}, \tag{5.14}$$

and

$$U(\beta,x,\theta_M,h) := \frac{\left[\left(x+\frac{2}{\beta}\sin 2\theta_M\right)\sin^2\theta_M-\cos^2\theta_M\right]}{2h}\check{U}$$
$$= \begin{pmatrix} \frac{\left(x+\frac{2}{\beta}\sin 2\theta_1\right)\sin^2\theta_1-\cos^2\theta_1}{2h} & & \\ & \ddots & \\ & & \frac{\left(x+\frac{2}{\beta}\sin 2\theta_M\right)\sin^2\theta_M-\cos^2\theta_M}{2h} \end{pmatrix}\check{U}, \tag{5.15}$$

where $\theta_M = [\theta_1,\theta_2,\cdots,\theta_M]^T$. Then

$$\frac{\partial \mathbf{H}_M}{\partial x} = \left[T(\beta,\theta_M,h)+U(\beta,x,\theta_M,h)\right]\mathbf{H}_M. \tag{5.16}$$

Noting that x is the time-like variable, we apply the trapezoidal rule with time step $\Delta x<0$ to (5.16) yielding,

$$\frac{\mathbf{H}_M^{n+1}-\mathbf{H}_M^n}{\Delta x} = \frac{1}{2}\left[T(\beta,\theta_M,h)\mathbf{H}_M^n+U(\beta,x_n,\theta_M,h)\mathbf{H}_M^n\right]+\frac{1}{2}\left[T(\beta,\theta_M,h)\mathbf{H}_M^{n+1}+U(\beta,x_{n+1},\theta_M,h)\mathbf{H}_M^{n+1}\right], \tag{5.17}$$

where $\mathbf{H}_M^n \approx \mathbf{H}_M(x_n)$. Upon rearranging, this gives,

$$\left[I-\frac{\Delta x}{2}T(\beta,\theta_M,h)-\frac{\Delta x}{2}U(\beta,x_{n+1},\theta_M,h)\right]\mathbf{H}_M^{n+1} = \left[I+\frac{\Delta x}{2}T(\beta,\theta_M,h)+\frac{\Delta x}{2}U(\beta,x_n,\theta_M,h)\right]\mathbf{H}_M^n. \tag{5.18}$$

Finally, we obtain $F_\beta(x)\approx\tilde{F}_\beta(x) := H_M(x)\approx H_M^n$, $n=0,1,\cdots,N$.

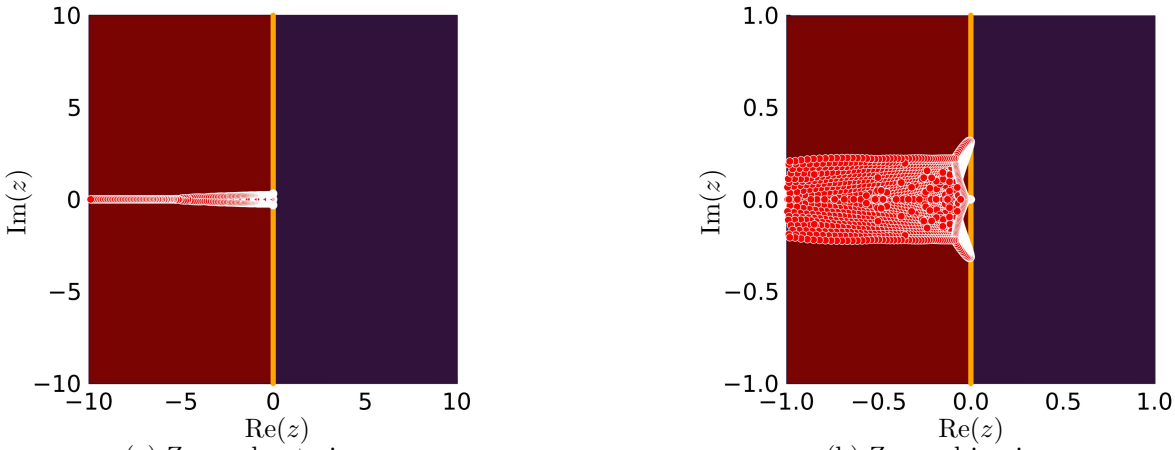
5.2.1 Eigenvalues of $\Delta x\left[T(\beta,\theta_M,h)+U(\beta,x,\theta_M,h)\right]$ for the FD discretization

The default time-stepping routine for the FD discretization is the trapezoidal method but BDF3, BDF4, BDF5, and BDF6 can also be used on (5.12) to solve for $\mathbf{H}_M$. It turns out that with values for the parameters as follows

$$x_0 = \left\lfloor\frac{13}{\sqrt{\beta}}\right\rfloor,\ x_N=-10,\ \Delta x=-10^{-3},\ \theta_M=\pi,\ M=10^3, \tag{5.19}$$

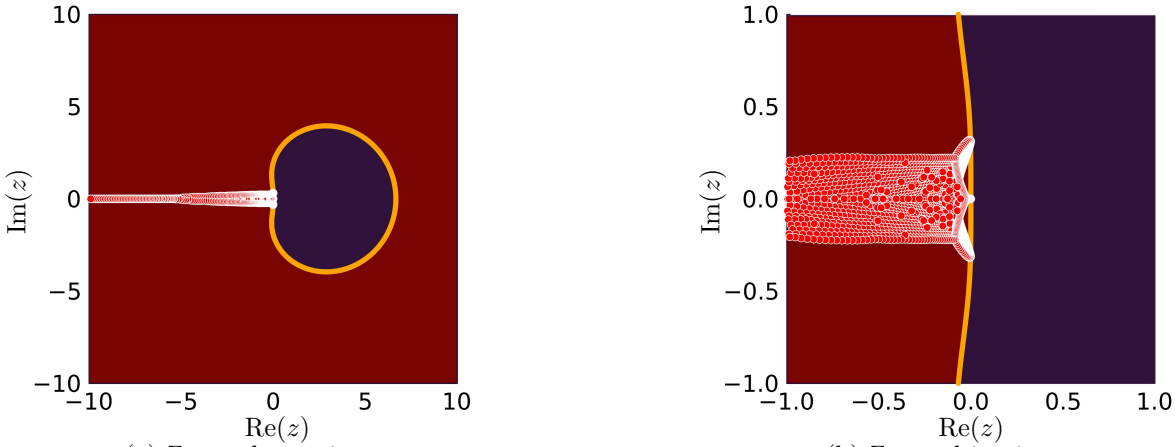
convergence occurs for all these methods. Since with the FD discretization, the trapezoidal method and BDF3 are usually used, Figure 5.1 and Figure 5.2 show the absolute stability regions of trapezoidal method and BDF3 along with eigenvalues of $\Delta x(T+U)$ for $\beta=2$, $x=x_0,x_0-1,\cdots,x_N$, $\Delta x=-10^{-3}$, and $M=10^3$. The

eigenvalues in the left-half plane are firmly within the region of absolute stability in each case.



(a) Zoomed-out view (b) Zoomed-in view

Figure 5.1: The absolute stability region of the trapezoidal method applied to the FD discretization (the maroon-shaded region along with the orange boundary curve) and the eigenvalues of $\Delta x\,(T+U)$ (the red dots with white boundary) for $\beta = 2$, $x = x_0, x_0 - 1, \cdots, x_N$, $\Delta x = -10^{-3}$, and $M = 10^3$.

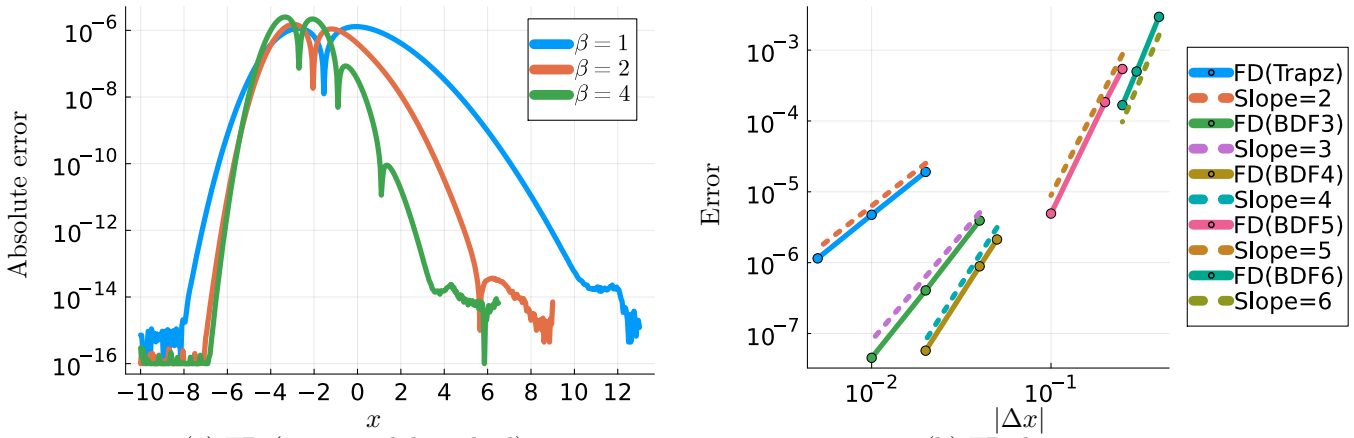


(a) Zoomed-out view (b) Zoomed-in view

Figure 5.2: The absolute stability region of the BDF3 applied to the FD discretization (the maroon-shaded region along with the orange boundary curve) and the eigenvalues of $\Delta x\,(T+U)$ (the red dots with white boundary) for $\beta = 2$, $x = x_0, x_0 - 1, \cdots, x_N$, $\Delta x = -10^{-3}$, and $M = 10^3$.

5.2.2 Order of the error

With values of the parameters given as in (5.19), Figure 5.3(a) shows the pointwise absolute errors of the FD method in the domain. We can see that the error is roughly on the order of 10^{-6} around the peak of the distribution and improves when x approaches either end of the domain. This is due to the fact that exact values, 0 and 1, for the initial condition are imposed on the endpoints of the domain, and the Dirichlet boundary condition ensures that the solution tends to zero.



(a) FD (Trapezoidal method) (b) FD discretization

Figure 5.3: (a) Pointwise absolute errors of the trapezoidal method are presented over the domain $x \in [-10, 13]$ for $\beta = 1, 2, 4$ with $x_0 = \lfloor 13/\sqrt{\beta} \rfloor$, $\Delta x = -10^{-3}$, and $M = 10^3$. (b) The order of accuracy for the FD discretization. The errors for the trapezoidal method are computed by treating the case with $\beta = 2$, $x_0 = \lfloor 13/\sqrt{\beta} \rfloor$, $\Delta x = -10^{-3}$, and $M = 10^3$ as the reference and compared with $\Delta x = -0.005, -0.01, -0.02$ at $x = -2$. For BDF methods, the same reference is used. For BDF3, $\Delta x = -0.01, -0.02, -0.04$ are used. For BDF4, $\Delta x = -0.02, -0.04, -0.05$ are used. For BDF5, $\Delta x = -0.1, -0.2, -0.25$ are used. For BDF6, $\Delta x = -0.25, -0.3, -0.4$ are used.

For $\beta = 2$ evaluated at $x = -2$, Figure 5.3(b) shows the order of error plots of the FD discretization. We choose $x = -2$ since it is near the peak of the distribution.

For the FD discretization, from Figure 5.3(a), the error has the expected order with respect to $|\Delta x|$. However, if we decrease the value of $|\Delta x|$ further, each BDF method has a corresponding range for Δx that causes instability. Moreover, once the value of Δx is out of this range, the corresponding BDF method becomes accurate again (see [TZ24] for more details).

5.3 Computing the Tracy-Widom distribution using a Fourier time-stepping method

To obtain better accuracy, we apply a Fourier spectral method. As before, for $0 \le n \le N$, define $x_n = x_0 + n\Delta x$. Suppose $H(\theta, x) = \int_0^\theta \rho(\theta', x)\, d\theta'$, then we rewrite (5.9) as

$$\frac{\partial H}{\partial x} + \left(\frac{2}{\beta}\sin^4\theta\right)\frac{\partial \rho}{\partial \theta} + \left[\left(x + \frac{2}{\beta}\sin 2\theta\right)\sin^2\theta - \cos^2\theta\right]\rho = 0. \tag{5.20}$$

Upon taking the derivative with respect to θ on both sides, we arrive at a partial differential equation for $\rho(\theta, x)$,

$$\begin{aligned}
&\frac{\partial \rho}{\partial x} + \left(\frac{8}{\beta}\sin^3\theta\cos\theta\right)\frac{\partial \rho}{\partial \theta} + \left(\frac{2}{\beta}\sin^4\theta\right)\frac{\partial^2 \rho}{\partial \theta^2} \\
&\quad + \left[(2x+2)\sin\theta\cos\theta + \frac{2}{\beta}\left(2\sin^2\theta\cos 2\theta + 2\sin\theta\cos\theta\sin 2\theta\right)\right]\rho \\
&\quad + \left[\left(x + \frac{2}{\beta}\sin 2\theta\right)\sin^2\theta - \cos^2\theta\right]\frac{\partial \rho}{\partial \theta} = 0.
\end{aligned} \tag{5.21}$$

Now, suppose

$$\rho(\theta, x) \approx \sum_{m=-M}^{M} a_m(x) e^{2im\theta/l}, \quad \theta \in [0, l\pi), \quad \theta_M = l\pi. \tag{5.22}$$

Substituting (5.22) in (5.21) gives a system of ordinary differential equations for $a_m(x)$, $m = -M, \cdots, M$, after truncation,

$$\frac{d\mathbf{a}_M(x)}{dx} = (A + xB)\,\mathbf{a}_M(x), \;\; \mathbf{a}_M(x) = \begin{pmatrix} a_{-M}(x) \\ \vdots \\ a_M(x) \end{pmatrix}, \tag{5.23}$$

where

$$\begin{aligned}
A = &-\frac{2}{\beta}\left(\frac{3}{8}I - \frac{1}{4}S_{-l} - \frac{1}{4}S_{+l} + \frac{1}{16}S_{+2l} + \frac{1}{16}S_{-2l}\right)D_2 \\
&+ \left[\frac{1}{2}I + \frac{1}{4}S_{-l} + \frac{1}{4}S_{+l} - \frac{2}{\beta}\left(\frac{1}{2i}S_{-l} - \frac{1}{2i}S_{+l}\right)\left(\frac{1}{2}I - \frac{1}{4}S_{+l} - \frac{1}{4}S_{-l}\right)\right]D_1 \\
&- \left[\frac{8}{\beta}\left(\frac{1}{-8i}S_{+3l/2} + \frac{1}{8i}S_{-3l/2} + \frac{3}{8i}S_{-l/2} - \frac{3}{8i}S_{+l/2}\right)\left(\frac{1}{2}S_{-l/2} + \frac{1}{2}S_{+l/2}\right)\right]D_1 \\
&- \frac{4}{\beta}\left(\frac{1}{2}I - \frac{1}{4}S_{+l} - \frac{1}{4}S_{-l}\right)\left(\frac{1}{2}S_{+l} + \frac{1}{2}S_{-l}\right) \\
&- \frac{4}{\beta}\left(\frac{1}{2i}S_{-l/2} - \frac{1}{2i}S_{+l/2}\right)\left(\frac{1}{2}S_{+l/2} + \frac{1}{2}S_{-l/2}\right)\left(\frac{1}{2i}S_{-l} - \frac{1}{2i}S_{+l}\right) \\
&- 2\left(\frac{1}{2i}S_{-l/2} - \frac{1}{2i}S_{+l/2}\right)\left(\frac{1}{2}S_{+l/2} + \frac{1}{2}S_{-l/2}\right),
\end{aligned} \tag{5.24}$$

and

$$B = \left(-\frac{1}{2}I + \frac{1}{4}S_{-l} + \frac{1}{4}S_{+l}\right)D_1 - 2\left(\frac{1}{2i}S_{-l/2} - \frac{1}{2i}S_{+l/2}\right)\left(\frac{1}{2}S_{+l/2} + \frac{1}{2}S_{-l/2}\right). \tag{5.25}$$

Here $S_{\pm k}$ represent the (Fourier modes) shift matrices, $S_{\pm k} = S_{\pm 1}^k$, where

$$S_{+1} = \begin{pmatrix} 0 & 1 & & & & 0 \\ & 0 & 1 & & & \\ & & 0 & 1 & & \\ & & & \ddots & \ddots & \\ & & & & 0 & 1 \\ 1 & & & & & 0 \end{pmatrix}, \quad S_{-1} = \begin{pmatrix} 0 & & & & & 1 \\ 1 & 0 & & & & \\ 0 & 1 & 0 & & & \\ & & \ddots & \ddots & & \\ & & & 1 & 0 & 0 \\ 0 & & & & 1 & 0 \end{pmatrix}.$$

D_1 and D_2 represent the first and second-order differentiation matrices in the Fourier space, i.e., $D_2 = D_1^2$, where

$$D_1 = \begin{pmatrix} -2iM/l & & & & \\ & -2i(M-1)/l & & & \\ & & \ddots & & \\ & & & 2i(M-1)/l & \\ & & & & 2iM/l \end{pmatrix}.$$

The Fourier coefficients of the initial condition are obtained via

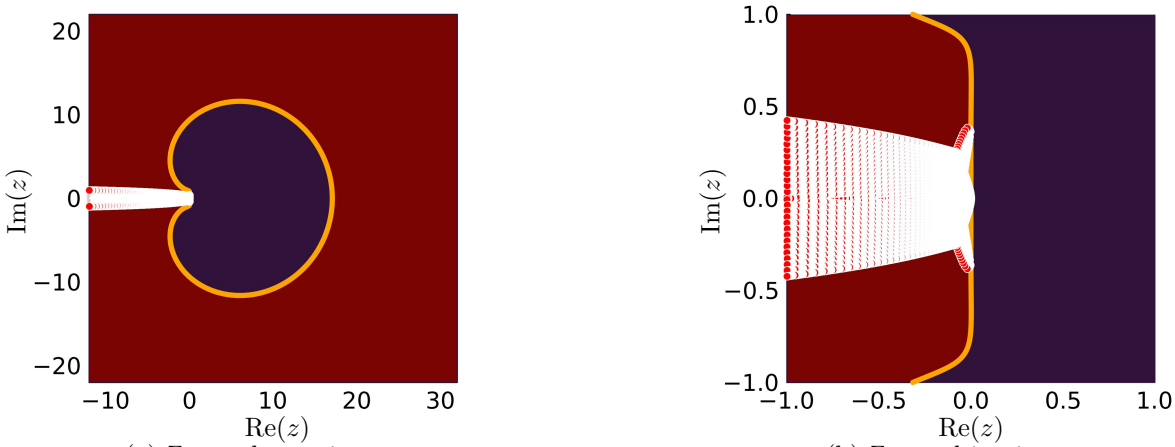
$$a_m(x_0) = \frac{1}{l\pi} \int_0^{l\pi} \rho(x_0, \theta) e^{-2im\theta/l} \, d\theta. \tag{5.26}$$

Instead of applying the second-order accurate trapezoidal rule to integrate the system of ODEs for $\rho(\theta, x)$, we suggest the use of BDF5, see [LeV07, Section 8.4]. To use BDF5, we need four more starting conditions, i.e., $\mathbf{a}_M(x_0 - i\Delta x)$, $i = 1, 2, 3, 4$, which can be obtained by (5.10) and (5.26). We then proceed with BDF5 to solve for $\mathbf{a}_M^n \approx \mathbf{a}_M(x_n)$

$$\left[137I - 60\Delta x \left(A + x_n B\right)\right] \mathbf{a}_M^n = 300\mathbf{a}_M^{n-1} - 300\mathbf{a}_M^{n-2} + 200\mathbf{a}_M^{n-3} - 75\mathbf{a}_M^{n-4} + 12\mathbf{a}_M^{n-5}. \tag{5.27}$$

Finally, the value of $H(\theta, x)$ can be recovered from

$$H(\theta, x) \approx \int_0^\theta \sum_{m=-M}^{M} a_m(x) e^{2im\theta'/l} \, d\theta' = \sum_{m=-M}^{M} a_m(x) \int_0^\theta e^{2im\theta'/l} \, d\theta'. \tag{5.28}$$



(a) Zoomed-out view (b) Zoomed-in view

Figure 5.4: The absolute stability region of the BDF5 applied to the Fourier spectral discretization (the maroon-shaded region along with the orange boundary curve) and the eigenvalues of $\Delta x\,(A + xB)$ (the red dots with white boundary) for $\beta = 2$, $x = x_0, x_0 - 1, \cdots, x_N$, $\Delta x = -10^{-3}$, and $M = 8 \times 10^3$.

The Tracy-Widom distribution can then be approximated by setting $\theta = \pi$,

$$F_\beta(x) \approx \tilde{F}_\beta(x) = \sum_{m=-M}^{M} a_m(x) \int_0^\pi e^{2im\theta'/l}\, d\theta' = \sum_{m=-M}^{M} a_m(x) \frac{l}{2im} \left(e^{2im\pi/l} - 1 \right), \tag{5.29}$$

from which we find

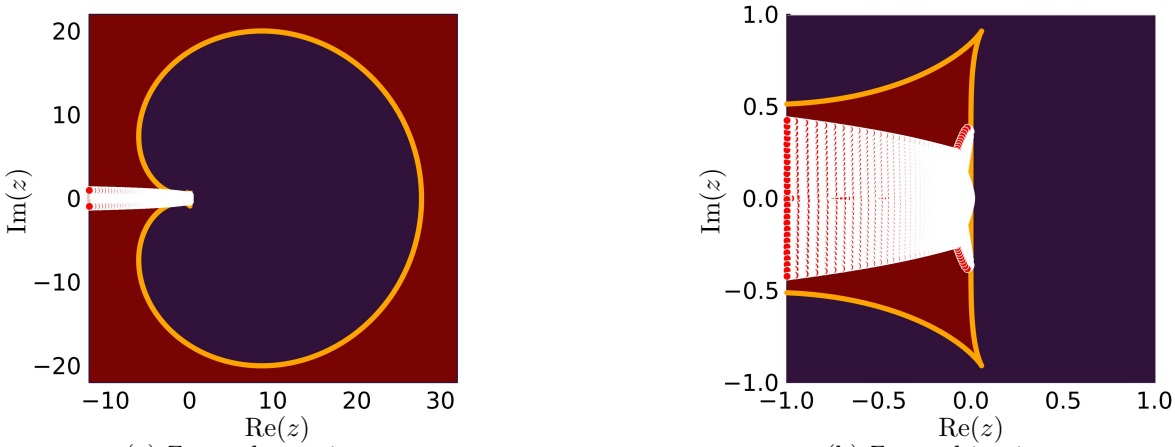
$$F_\beta(x_n) \approx \tilde{F}_\beta(x_n) \approx \sum_{m=-M}^{M} a_m^n \frac{l}{2im} \left(e^{2im\pi/l} - 1 \right), \quad n = 0, 1, \cdots, N. \tag{5.30}$$

5.3.1 Eigenvalues of $\Delta x\,(A + xB)$ for the Fourier spectral discretization

Figure 5.4 and Figure 5.5 show the absolute stability regions of BDF5 and BDF6 along with the eigenvalues of $\Delta x(A + xB)$ for $\beta = 2$, $x = x_0, x_0 - 1, \cdots, x_N$, $\Delta x = -10^{-3}$, and $M = 8 \times 10^3$. It is clear that, with the values of the parameters given as follows:

$$x_0 = \left\lfloor \frac{13}{\sqrt{\beta}} \right\rfloor, \; x_N = -10, \; \Delta x = -10^{-3}, \; \theta_M = 20\pi, \; M = 8 \times 10^3, \tag{5.31}$$

the eigenvalues in the left-half plane are firmly within the region of absolute stability for these methods. We choose BDF5 over BDF6 because with the same values of these parameters, the performance is roughly the same yet BDF5 has a larger absolute stability region. Selecting Δx is more nuanced than one might think (see [TZ24] for more details).

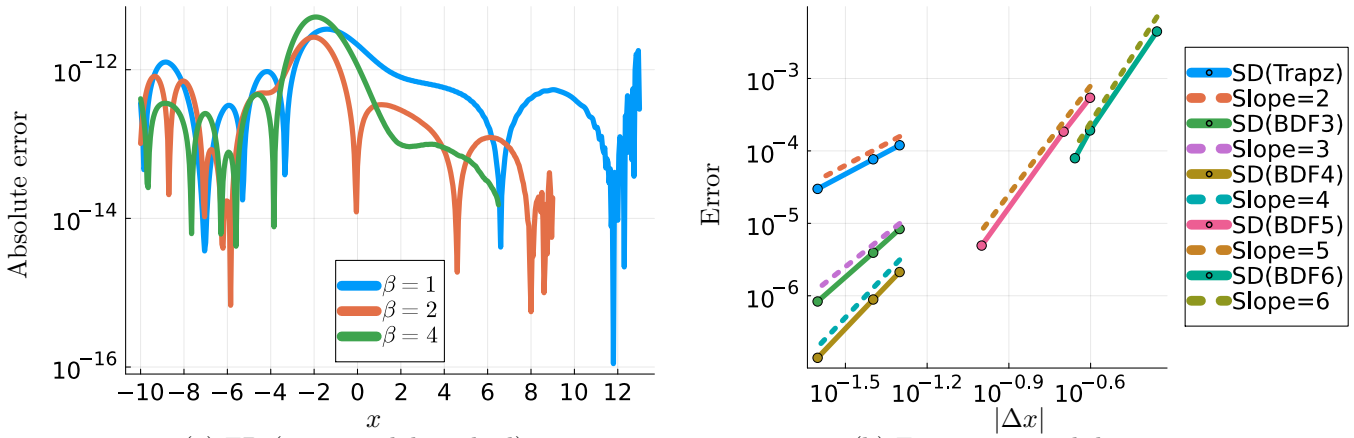


(a) Zoomed-out view (b) Zoomed-in view

Figure 5.5: The absolute stability region of the BDF6 applied to the Fourier spectral discretization (the maroon-shaded region along with the orange boundary curve) and the eigenvalues of $\Delta x\,(A + xB)$ (the red dots with white boundary) for $\beta = 2$, $x = x_0, x_0 - 1, \cdots, x_N$, $\Delta x = -10^{-3}$, and $M = 8 \times 10^3$.

5.3.2 Order of the error

With values of the parameters given as in (5.31), Figure 5.6(a) shows the pointwise absolute errors across the domain. We can see that the errors are roughly on the order of 10^{-13} in the domain. Regardless of the order of the error, the reason for this discrepancy from the FD discretization near the endpoints of the domain is that an error is introduced for the initial condition when we represent $\rho(\theta, x)$ in terms of a truncated Fourier series. For $\beta = 2$ evaluated at $x = -2$, Figure 5.6(b) shows the order of error plots of the spectral discretization. We can see that the error has the expected order with respect to $|\Delta x|$.



(a) FD (Trapezoidal method)

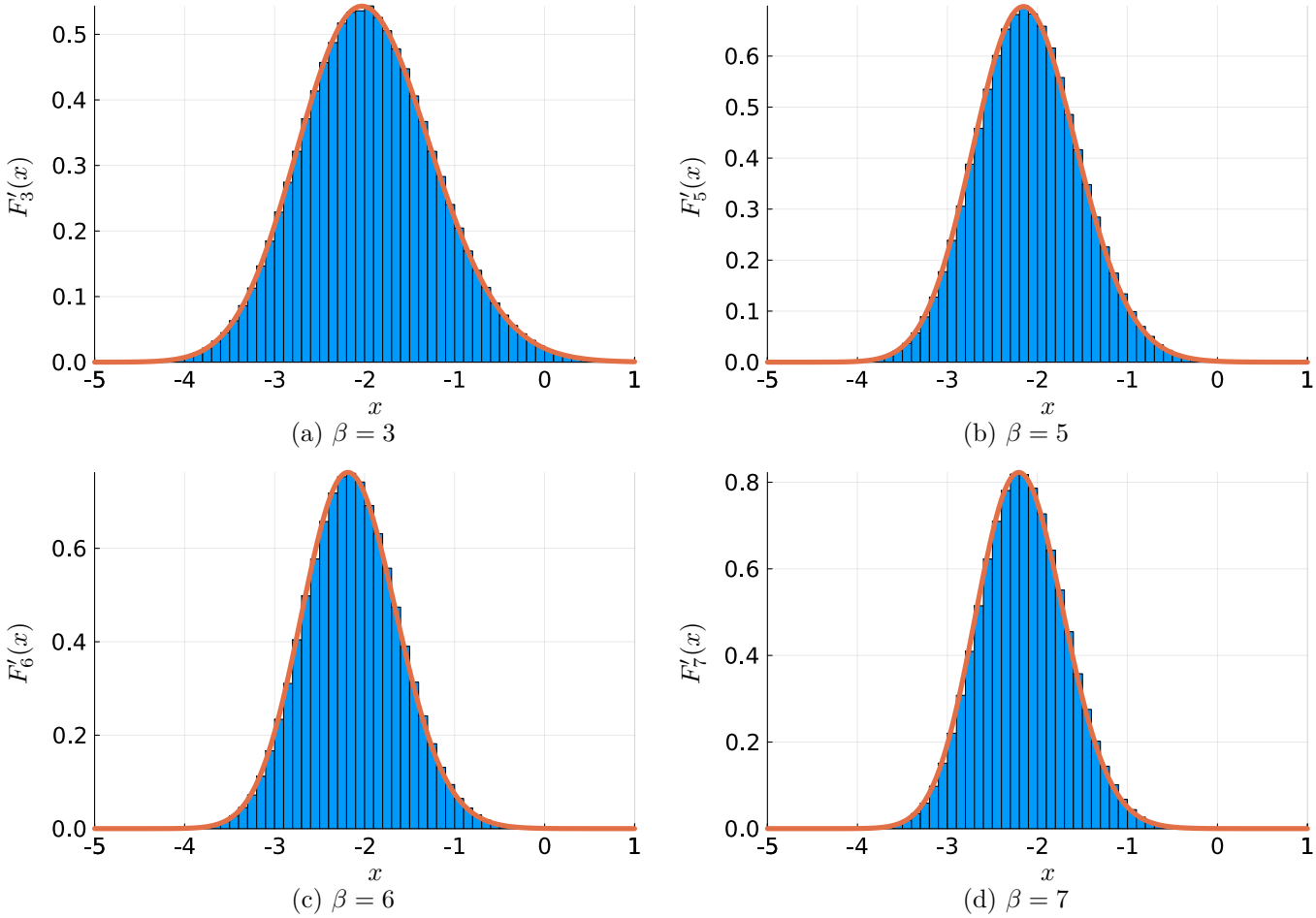
(b) Fourier spectral discretization

Figure 5.6: (a) Pointwise absolute errors of the Fourier time-stepping method are presented over the domain $x \in [-10, 13]$ for $\beta = 1, 2, 4$ with $x_0 = \lfloor 13/\sqrt{\beta} \rfloor$, $\Delta x = -10^{-3}$, and $M = 8 \times 10^3$. (b) The order of accuracy for the Fourier spectral discretization. The errors are computed by treating the case with $\beta = 2$, $x_0 = \lfloor 13/\sqrt{\beta} \rfloor$, $\Delta x = -10^{-3}$, and $M = 8000$ as the reference and compared with $\Delta x = -0.025, -0.04, -0.05$ for the trapezoidal method at $x = -2$. For BDF3 and BDF4, $\Delta x = -0.025, -0.04, -0.05$ are used. For BDF5, $\Delta x = -0.1, -0.2, -0.25$ are used. For BDF6, $\Delta x = -0.22, -0.25, -0.44$ are used.

5.4 Additional numerical results

In this section, we present some additional plots that are related to the Tracy-Widom distribution.

5.4.1 Comparison with large random matrices



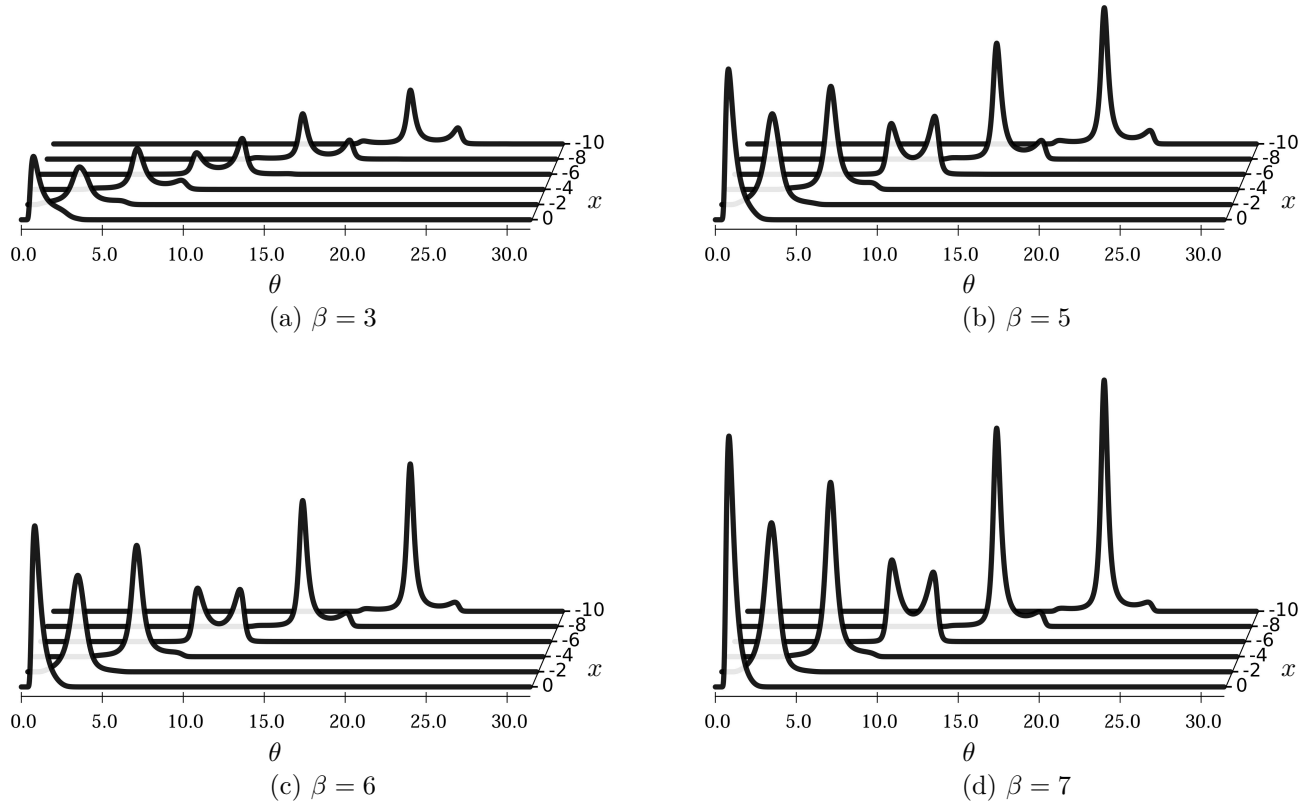
(a) $\beta = 3$ (b) $\beta = 5$ (c) $\beta = 6$ (d) $\beta = 7$

Figure 5.7: Histograms for $n^{1/6}\left(\lambda_{\max}(H_n^\beta) - 2\sqrt{n}\right)$ using 10^6 samples with $n = 10^4$. The density $F_\beta'(x)$ is generated by the FD method for $\beta = 3, 5, 6, 7$. The small positive bias in each histogram can be improved by using larger values of n.

We verify numerically that the pdf generated by the FD method agrees with the model presented by Dumitriu and Edelman [DE02]. Recall that H_n^β in (5.3) has (5.2) as the jpdf for its eigenvalues, and the distribution of its largest eigenvalue, after rescaling, converges to F_β for any $\beta > 0$ as $n \to \infty$.

Histograms in Figure 5.7 are the normalized histograms for $n^{1/6}\left(\lambda_{\max}(H_n^\beta) - 2\sqrt{n}\right)$, where $\lambda_{\max}(H_n^\beta)$ denotes the largest eigenvalue of the β-Hermite ensemble H_n^β.

5.4.2 Evolution of the density $\rho(\theta, x) := \partial H(\theta, x)/\partial\theta$

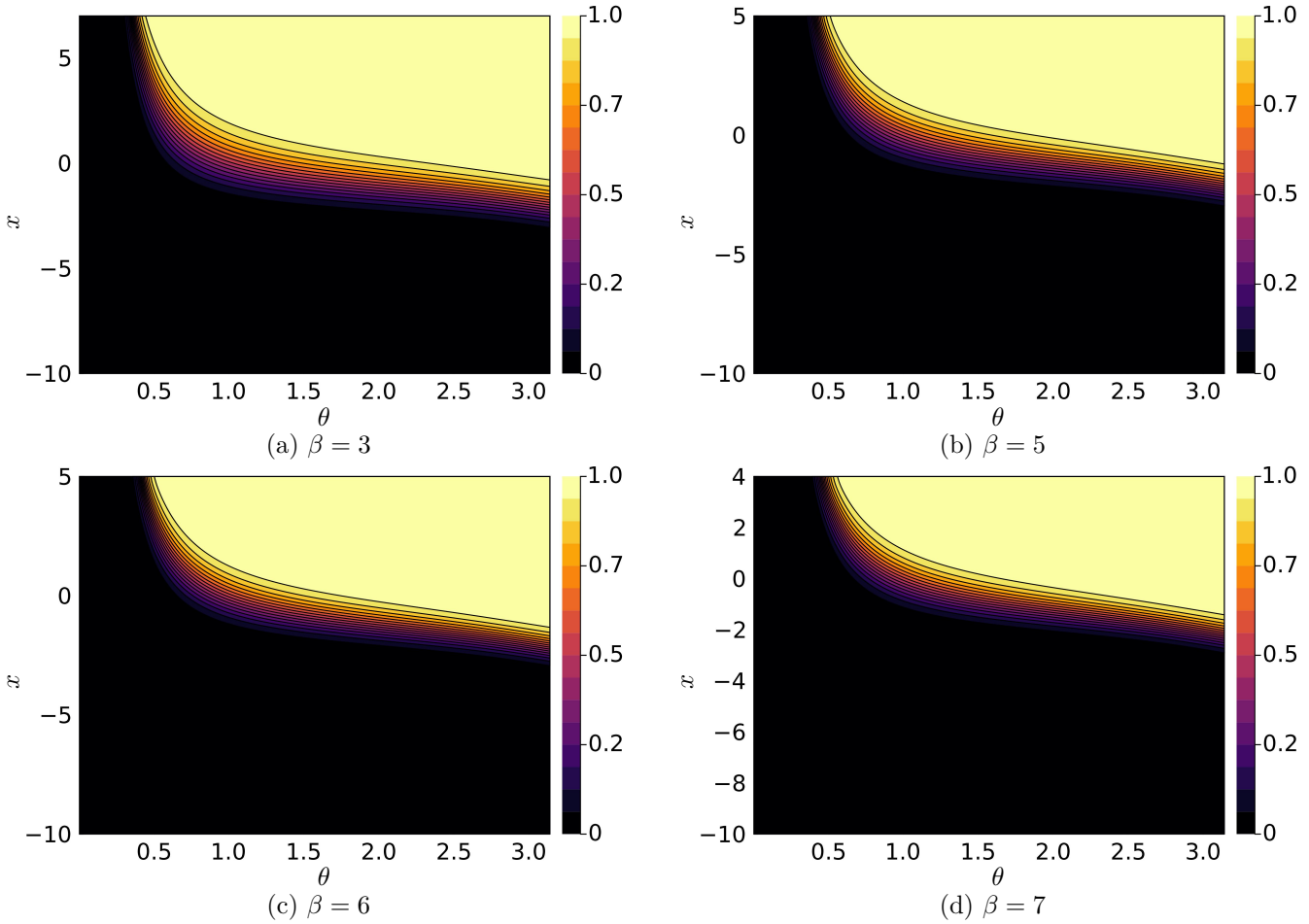


(a) $\beta = 3$ (b) $\beta = 5$

(c) $\beta = 6$ (d) $\beta = 7$

Figure 5.8: The evolution of $\rho(\theta, x) := \frac{\partial H}{\partial \theta}$ as x decreases from $x = 0$ to -10 with step size $= -2$ for $\beta = 3, 5, 6, 7$. FD discretization with trapezoidal method is used on (5.21) with $\Delta x = -10^{-3}$ and $M = 10^4$ for $\theta \in [0, 10\pi]$.

Figure 5.8 shows the waterfall plots of the density $\rho(\theta, x) := \partial H(\theta, x)/\partial\theta$ for $\theta \in [0, 10\pi]$ and $x = 0, -2, \cdots, -10$ using the FD discretization with trapezoidal method on (5.21) with $\theta_M = 10\pi$ and $M = 10^4$. The values of the other parameters are the same as in (5.19). As the value of x decreases, the density of the solution propagates to the right.



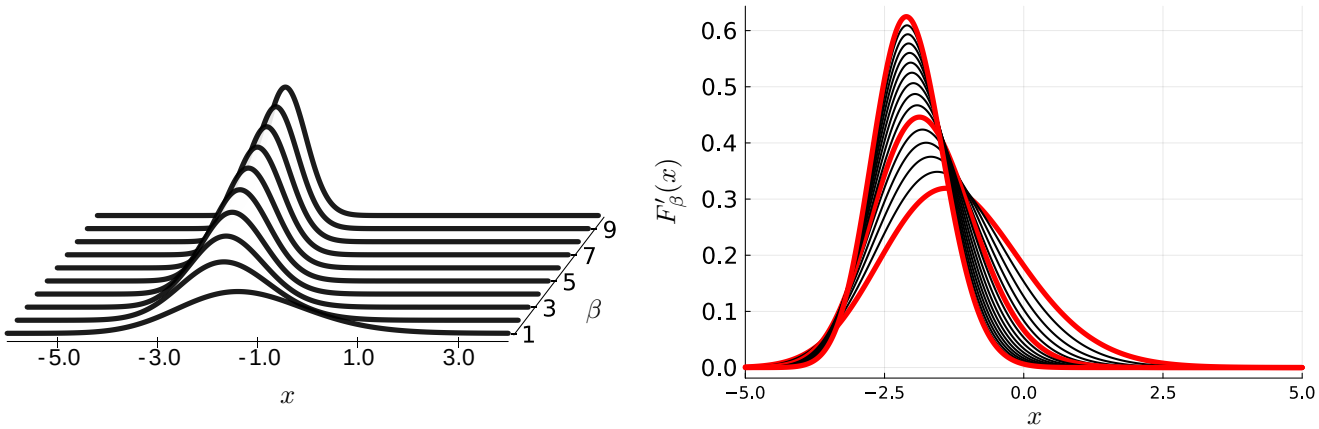
(a) $\beta = 3$ (b) $\beta = 5$

(c) $\beta = 6$ (d) $\beta = 7$

Figure 5.9: The evolution of $H(\theta, x)$ as x decreases from $\lfloor 13/\sqrt{\beta} \rfloor$ to -10 for $\beta = 3, 5, 6, 7$. FD discretization with trapezoidal method is used with $\Delta x = -10^{-3}$ and $M = 10^3$.

Figure 5.9 shows the contour plots of the approximation of $H(\theta, x)$ using the FD discretization with trapezoidal method with values of the parameters given as in (5.19). For each contour plot, the initial condition $H(x_0, \theta)$ is found along the top of the plot and the approximate Tracy-Widom distribution, $F_\beta(x) \approx H(\theta_M, x)$, is obtained on the right-hand side of the plot.

5.4.3 Density of the Tracy-Widom distribution



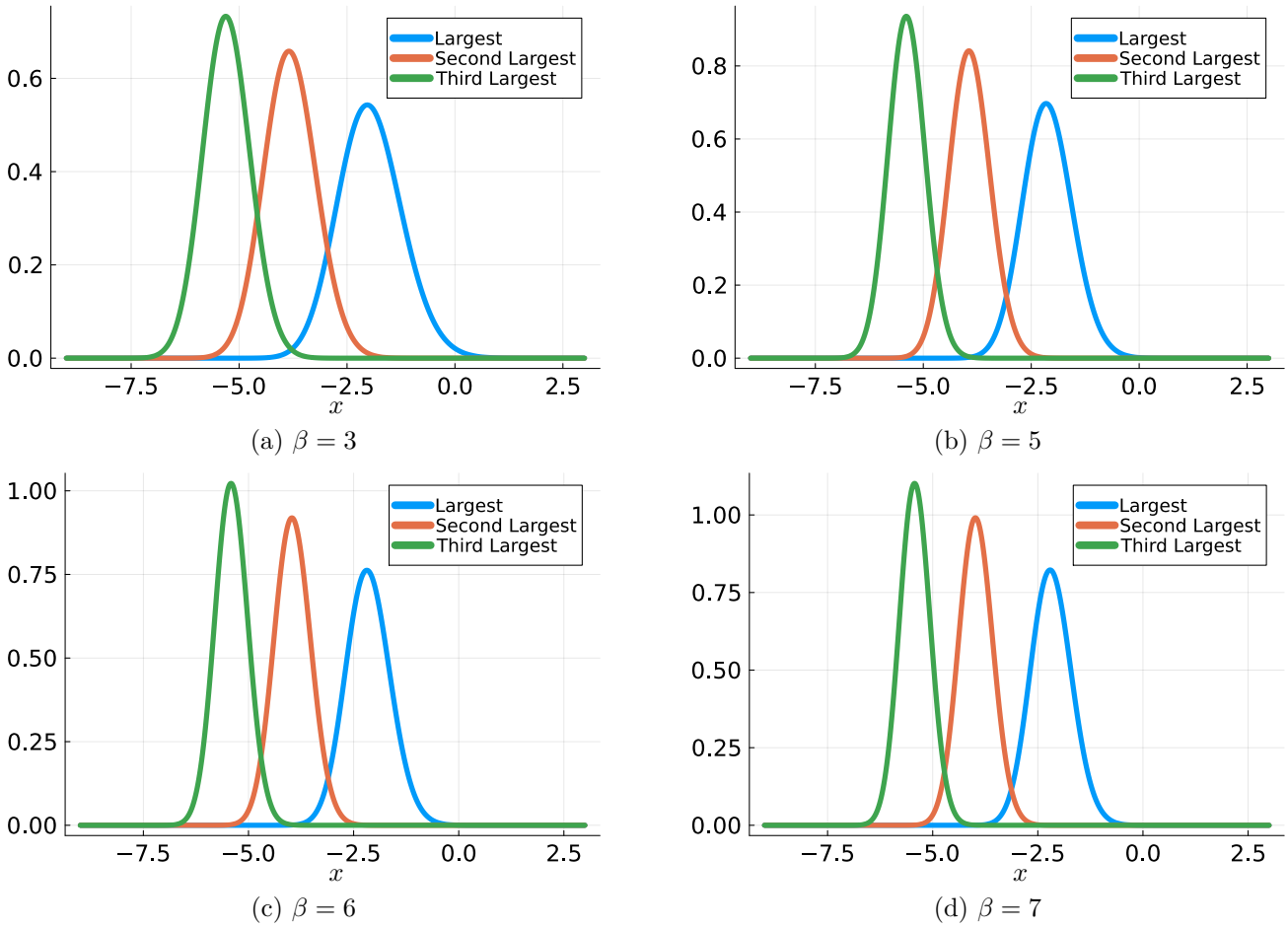
(a) $\beta = 1$ to 10

(b) $\beta = 1$ to 4

Figure 5.10: PDFs of the Tracy-Widom distribution for different values of β using the FD discretization with trapezoidal method with values for the other parameters as in (5.19).

Figure 5.10(a) shows the plot of the approximation of F'_β for $\beta = 1$ to 10 and $x \in [-6, 4]$ using the FD discretization with the trapezoidal method with values of the other parameters as in (5.19). As we can see from the plot, as the value of β increases, F'_β becomes more concentrated, and its peak moves leftwards. Figure 5.10(b) is a two-dimensional version of Figure 5.10(a), which is also generated using the FD discretization with the trapezoidal method. It provides a closer view of F'_β for $\beta = 1$ to 4 with step size $= 0.2$. The red curves from right to left correspond to $\beta = 1, 2, 4$ respectively. The black curves show F'_β for the other values of β.

5.4.4 Limiting densities of the other eigenvalues



(a) $\beta = 3$ (b) $\beta = 5$

(c) $\beta = 6$ (d) $\beta = 7$

Figure 5.11: Densities of $-\Lambda_0$, $-\Lambda_1$, and $-\Lambda_2$ of $\mathcal{H}_\beta$ for $\beta = 3, 5, 6, 7$. Finite-difference discretization with trapezoidal method is used with values of the parameters as in (5.19).

By [Blo11, Theorem 2.4.3], one finds the limiting density of the kth largest eigenvalue, after rescaling, of the β-Hermite ensemble H_n^β at $\theta = k\pi$. Using the FD discretization with the trapezoidal method with values for the parameters as in (5.19), Figure 5.11 shows the limiting densities of the largest three eigenvalues of H_n^β, namely, $F'(\cdot, \pi)$, $F'(\cdot, 2\pi)$, and $F'(\cdot, 3\pi)$ from right to left respectively. They can also be interpreted as the densities of $-\Lambda_0$, $-\Lambda_1$, and $-\Lambda_2$ of the stochastic Airy operator $\mathcal{H}_\beta$. As the value of k increases, the limiting distribution becomes more concentrated.

5.5 Computing the Tracy-Widom distribution using the spectral time-stepping method

By representing the solution in terms of Chebyshev series in both spatial and temporal variables, this problem will be solved over $x \in [x_e, x_0]$ and $\theta \in [0, \pi]$. Since Chebyshev polynomials are defined over $[-1, 1]$, we need to perform the following transformations:

$$\hat{t} := \psi(x) = \frac{2(x - x_e)}{(x_0 - x_e)} - 1, \quad \hat{x} := \phi(\theta) = \frac{2\theta}{\pi} - 1. \tag{5.32}$$

In this way, (5.9) becomes $\mathcal{L}H = 0$, where the PDO $\mathcal{L}$ is given by

$$\mathcal{L} = \frac{2}{x_0 - x_e}\frac{\partial}{\partial t} + \frac{8}{\beta\pi^2}\sin^4\left[\frac{\pi\,(t+1)}{2}\right]\frac{\partial^2}{\partial t^2} + \left\{\left(\frac{x_0 - x_e}{2}\,(t+1) + x_e + \frac{2}{\beta}\sin\left[\pi\,(t+1)\right]\right) \sin^2\left[\frac{\pi\,(t+1)}{2}\right] - \cos^2\left[\frac{\pi\,(t+1)}{2}\right]\right\}\frac{2}{\pi}\frac{\partial}{\partial t}. \tag{5.33}$$

The splitting rank of $\mathcal{L}$ is 3 since we have the following decomposition:

$$\mathcal{L} = \mathcal{D} \otimes \frac{2}{x_0 - x_e}\mathcal{J} + \mathcal{J} \otimes \left\{\frac{8}{\beta\pi^2}\sin^4\left[\frac{\pi\,(t+1)}{2}\right]\mathcal{D}^2 + \left(\frac{2}{\beta}\sin\left[\pi\,(t+1)\right]\sin^2\left[\frac{\pi\,(t+1)}{2}\right] - \cos^2\left[\frac{\pi\,(t+1)}{2}\right]\right)\frac{2}{\pi}\mathcal{D}\right\} + \left[\frac{x_0 - x_e}{2}\,(t+1) + x_e\right]\mathcal{J} \otimes \frac{2}{\pi}\sin^2\left[\frac{\pi\,(t+1)}{2}\right]\mathcal{D}, \tag{5.34}$$

where $\mathcal{J}$ is the identity operator and $\mathcal{D}$ is the 1st-order differential operator.

In the new domain $(\hat{x}, \hat{t}) \in [-1, 1] \times [-1, 1]$, we want to solve $\mathcal{L}H(\hat{x}, \hat{t}) = 0$. $\mathcal{L}$ can be decomposed into Kronecker products of ODOs, which can be truncated to derive the generalized Sylvester matrix equation $A_1 X C_1^T + A_2 X C_2^T + A_3 X C_3^T = F$, where

$$A_1 = \mathcal{P}_{n_t}\mathcal{D}_1\mathcal{P}_{n_t}^T, \quad C_1 = \mathcal{P}_{n_x}\frac{2}{x_0 - x_e}\mathcal{S}_1\mathcal{S}_0\mathcal{P}_{n_x}^T, \tag{5.35}$$

$$A_2 = \mathcal{P}_{n_t}\mathcal{S}_0\mathcal{P}_{n_t}^T, \quad C_2 = \mathcal{P}_{n_x}\left(\frac{8}{\beta\pi^2}\mathcal{M}_2\left\{\sin^4\left[\frac{\pi\,(t+1)}{2}\right]\right\}\mathcal{D}_2 + \frac{2}{\pi}\mathcal{S}_1\mathcal{M}_1\left\{\frac{2}{\beta}\sin\left[\pi(t+1)\right]\sin^2\left[\frac{\pi\,(t+1)}{2}\right] - \cos^2\left[\frac{\pi\,(t+1)}{2}\right]\right\}\mathcal{D}_1\right)\mathcal{P}_{n_x}^T, \tag{5.36}$$

$$A_3 = \mathcal{P}_{n_t}\mathcal{S}_0\mathcal{M}_0\left[\frac{x_0 - x_e}{2}(t+1) + x_e\right]\mathcal{P}_{n_t}^T, \quad C_3 = \mathcal{P}_{n_x}\frac{2}{\pi}\mathcal{S}_1\mathcal{M}_1\left\{\sin^2\left[\frac{\pi(t+1)}{2}\right]\right\}\mathcal{D}_1\mathcal{P}_{n_x}^T, \tag{5.37}$$

and F is the $n_t \times n_x$ matrix of Chebyshev coefficients for the right-hand side function, which is just the zero matrix. For the boundary conditions, we have $XB_x^\intercal = G^\intercal$, where $B_x = \big(T_0(-1), T_1(-1), \cdots, T_{n_{x-1}}(-1)\big)$, and G is an $1 \times n_t$ zero matrix. Similarly, on the top boundary, we have $B_t X = K$, where $B_t = \big(T_0(1), T_1(1), \cdots, T_{n_{t-1}}(1)\big)$, and K is an $1 \times n_x$ matrix containing the first n_x Chebyshev coefficients of the initial condition. By enforcing the boundary conditions in the way described in Section 4.3, we will need to solve (4.17) for $k = 3$. The `Julia` code used to solve this problem for $\beta = 1, 2, 4$ using a 10-step spectral time-stepping method is as follows with `x0` $= \lfloor 13/\sqrt{\beta} \rfloor$, `xe` $= -10$, and `N` $= 10$. The package `ApproxFun.jl` [OGS+15] is used.

```
Z1 = spzeros(1, n_t); Z2 = spzeros(1, n_x); Z3 = spzeros(n_t-1, n_t); Z4 = spzeros(n_x-1, n_x)
idx = sparse(I,n_x,n_x); idt = sparse(I,n_t,n_t); cheb_coeffr = spzeros(n_t, N)
Phi = x -> erf.(x/sqrt(2))/2 + 0.5; g = (t) -> Phi( (x0 - cot(t).^2)./sqrt.(4/beta * cot.(t)) )
h0 = (t) -> t < 0 ? 0 : (0 <= t < π/2 ? g(t) : 1)
d = 0..π; cheb_space = Chebyshev(d); cheb_piecewise_gau = Fun(t -> h0(t), cheb_space, n_x)
coeffb = cheb_piecewise_gau.coefficients
s = (x0 - xe)/N; endx = [x0 - i*s for i in 0:N]; F = spzeros(n_t,n_x); Bt = ones(1,n_t)
BBt = vcat(Bt,Z3); Bx = ones(1,n_x); Bx = [(-1.0)^(i-1) for i in 1:n_x]; Bx = Bx'; BBx = vcat(Bx,Z4)
D_1 = Derivative(); D_1 = D_1:Chebyshev(-1..1); D_2 = (D_1)^2; v = [(-1)^(i-1) for i in 1:n_t]
S_02 = Conversion(Chebyshev(-1..1),Ultraspherical(2))
C_1 = dropzeros!(sparse((2/s)*S_02[1:n_x-1,1:n_x])); CC1 = vcat(Z2,C_1); x = Fun(-1..1)
M1_x2 = Multiplication((2/beta)*sin(π*(x+1))*(sin(π*(x+1)/2))^2
-(cos(π*(x+1)/2))^2,Ultraspherical(1))
M2_x2 = Multiplication((sin(π*(x+1)/2))^4, Ultraspherical(2))
S_12 = Conversion(Ultraspherical(1), Ultraspherical(2))
C_2 = dropzeros!(sparse((8/(beta*π^2))*M2_x2[1:n_x-1,1:n_x]*D_2[1:n_x,1:n_x]+(2/π)
*S_12[1:n_x-1,1:n_x]*M1_x2[1:n_x,1:n_x]*D_1[1:n_x,1:n_x])); CC2 = vcat(Z2,C_2)
M1_x3 = Multiplication((sin(π*(x+1)/2))^2, Ultraspherical(1))
C_3 = dropzeros!(sparse((2/π)*S_12[1:n_x-1,1:n_x]*M1_x3[1:n_x,1:n_x]*D_1[1:n_x,1:n_x]))
CC3 = vcat(Z2,C_3)
for j = 1:N
    x0 = endx[j]; xe = endx[j+1]; coefft = coeffb
    A_1 = dropzeros!(sparse(D_1[1:n_t-1,1:n_t])); AA1 = vcat(Z1,A_1)
    S_01 = Conversion(Chebyshev(-1..1), Ultraspherical(1))
    A_2 = dropzeros!(sparse(S_01[1:n_t-1,1:n_t])); AA2 = vcat(Z1,A_2)
```

```
    x2 = Fun(-1..1); M0_x = Multiplication((x0-xe)*(x2+1)/2+xe, Chebyshev())
    A_3 = dropzeros!(sparse(S_01[1:n_t-1,1:n_t]*M0_x[1:n_t,1:n_t])); AA3 = vcat(Z1,A_3)
    F[1,:] = coefft; Kr1 = kron(CC1,AA1)+kron(CC2,AA2)+kron(CC3,AA3)
    bl,bu = bandwidths(Kr1); Kr = Kr1 + kron(idx,BBt)+kron(BBx,idt)
    k = findfirst(i -> all(iszero, kron(BBx,idt)[i, :]), 1:size(kron(BBx,idt),1)) - 1
    B3,Ub,V = generate_data2(copy(L), n_t*n_x, vec(F), k, bu)
    Xb = woodbury_sparseqr(B3, Ub, V, k, bl); xn = reshape(Xb,n_t,n_x)
    coeffr = vec(sum(xn, dims=2)); cheb_coeffr[:,j] = coeffr; coeffb = vec(transpose(xn)*v)
end
x0 = floor(13/beta^(1/3))
TW_cdf = 0; TW_cdf = sum([Fun(x0-i*s..x0-(i-1)*s, cheb_coeffr[:,i]) for i in 1:N])
```

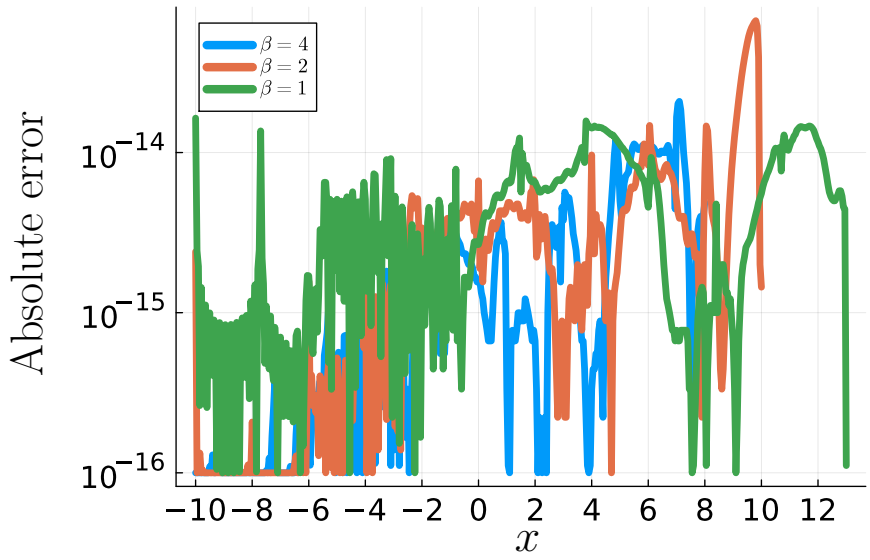


Figure 5.12: Pointwise absolute errors for $\beta = 1, 2, 4$ computed using a 10-step spectral time-stepping method. For $\beta = 1$, the errors are computed with 461 uniform grid points over $[-10, 13]$ with $n_x = 447$ and $n_t = 18$. For $\beta = 2$, the errors are computed with 401 uniform grid points over $[-10, 10]$ with $n_x = 395$ and $n_t = 20$. For $\beta = 4$, the errors are computed with 361 uniform grid points over $[-10, 8]$ with $n_x = 401$ and $n_t = 22$.

Figure 5.12 shows the pointwise absolute errors for $\beta = 1, 2, 4$, where the reference solutions are computed with high precision using Mathematica. Attempting to obtain these results via a single global spectral-in-time solve encounters issues such as insufficient precision or excessive memory requirements. Moreover, a single global solve incurs substantially greater computational complexity than the proposed N-step time-stepping approach for $N \geq 2$.

Chapter 6

Spectral-in-time methods for solving nonlinear time-evolution PDEs

To apply the spectral-in-time method to a nonlinear PDE defined on the domain $U_x \times [t_0, t_M]$, we reconsider the same PDE problem as in (4.1) subject to the boundary conditions in (4.2), but now introduce a nonlinear source term $F\left(t, x, u, \frac{\partial u}{\partial x}, \cdots, \frac{\partial^m u}{\partial x^m}\right)$ on the right-hand-side of (4.1), a nonlinear function of $u(x,t)$ and its spatial derivatives up to order m. Here, we assume that F does not depend on the temporal derivatives of $u(x,t)$. Similar to the nonlinear ODE problems discussed previously in Section 3.4, we use Newton's method for solving the nonlinear PDEs. Defining the nonlinear operator as

$$\mathcal{N}\left(t, x, u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}, \cdots, \frac{\partial^N u}{\partial x^N}\right) = \mathcal{L}u(x,t) - f(x,t) - F\left(t, x, u, \frac{\partial u}{\partial x}, \cdots, \frac{\partial^m u}{\partial x^m}\right), \tag{6.1}$$

Newton's iteration proceeds by solving the linearized approximation

$$\mathcal{N}\left(t, x, \frac{\partial u}{\partial t}, u + \delta u, \frac{\partial u}{\partial x} + \frac{\partial \delta u}{\partial x}, \cdots, \frac{\partial^N u}{\partial x^N} + \frac{\partial^N \delta u}{\partial x^N}\right) \approx \mathcal{N}\left(t, x, u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}, \cdots, \frac{\partial^N u}{\partial x^N}\right) + \mathcal{J}\delta u = 0, \tag{6.2}$$

where $\mathcal{J}$ is the Jacobian operator. To initialize Newton's method, we compute an initial guess $u^{(0)}(x,t)$ by solving the linear PDE $\mathcal{L}u^{(0)}(x,t) = f(x,t)$, subject to the boundary conditions $\mathcal{B}_x u^{(0)}(x,t) = \mathbf{g}(t)$ and $\mathcal{B}_t u^{(0)}(x,t) = \mathbf{h}(x)$. We then iterate for $k = 0, 1, 2, \ldots$ by solving the linearized equation for the increment $\delta u^{(k)}$:

$$\left(\mathcal{L} - \sum_{j=0}^{m} \frac{\partial^j F}{\partial u_{jx}^{(k)}} \mathcal{D}_j\right) \delta u^{(k)} = f(x,t) - \mathcal{L}u^{(k)} + F\left(t, x, u^{(k)}, \frac{\partial u^{(k)}}{\partial t}, \frac{\partial u^{(k)}}{\partial x}, \cdots, \frac{\partial^m u^{(k)}}{\partial x^m}\right), \tag{6.3}$$

subject to the boundary conditions

$$\mathcal{B}_x \delta u^{(k)}(x,t) = \mathbf{0}, \quad \mathcal{B}_t \delta u^{(k)}(x,t) = \mathbf{0}, \tag{6.4}$$

until the convergence criterion $\|\delta \mathbf{u}^{(k)}\| \leq \epsilon$ is satisfied for a given tolerance ϵ, where $\mathcal{D}_k$ denotes the kth-order spatial differentiation operator. Here, $\delta \mathbf{u}^{(k)}$ denotes the coefficient matrix associated with $\delta u^{(k)}$. If the

convergence criterion is not satisfied, we update the solution estimate as $u^{(k+1)}(x,t) = u^{(k)}(x,t) + \delta u^{(k)}(x,t)$ and continue the iteration. Once the convergence criterion is met at iteration $k = K$, the final solution is represented by the coefficient matrix $\mathbf{u} = \mathbf{u}^{(K)}$, and the solution is then approximated using (4.24), (4.26), or (4.32).

6.1 The Final-time Jacobian matrix method

However, unlike nonlinear ODEs, the solution u now depends on both spatial and temporal variables. Therefore, the resulting Jacobian matrix will be dense. Here, we present one way to represent one of the nonlinear terms of the Jacobian matrix on the left-hand side of (6.3).

Suppose that the coefficient matrix of the iterate $u^{(k)}(x,t)$ is given by X. We construct conversion matrices $T_t \in \mathbb{C}^{n_t \times n_t}$ and $T_x \in \mathbb{C}^{n_x \times n_x}$ that convert coefficients into grid values in the domain. We can then form the grid value matrix $Y_0 = T_t X T_x^{\mathsf{T}}$ of the iterate $u^{(k)}(x,t)$. If the kth-order spatial derivative of $u^{(k)}(x,t)$ is involved, we form the differentiated grid value matrix $Y_k = T_t X D_k^{\mathsf{T}} T_x^{\mathsf{T}} = T_t X \left(T_x D_k\right)^{\mathsf{T}}$. Then we apply the nonlinear function entry-wise to get the grid value matrix $f\left(Y_0, \cdots, Y_N\right)$. To multiply the grid value matrix of $\delta u^{(k)}$ entry-wise by $f\left(Y_0, \cdots, Y_N\right)$, we first convert it to a vector and put it on the diagonal of a diagonal matrix to get $\operatorname{diag}\left(\operatorname{vec}\left(f\left(Y_0, \cdots, Y_N\right)\right)\right)$. Then we can apply it to the grid value matrix of $\delta u^{(k)}$ through $\operatorname{diag}\left(\operatorname{vec}\left(f\left(Y_0, \cdots, Y_N\right)\right)\right)\left(T_x \otimes T_t\right)$. If we want to multiply the kth-order spatial derivative of $\delta u^{(k)}$ entry-wise, we do $\operatorname{diag}\left(\operatorname{vec}\left(f\left(Y_0, \cdots, Y_N\right)\right)\right)\left(T_x D_k \otimes T_t\right)$ instead. Finally, we convert this grid value matrix to multiplication matrix in the coefficient space as $\left(T_x^{-1} \otimes T_t^{-1}\right) \operatorname{diag}\left(\operatorname{vec}\left(f\left(Y_0, \cdots, Y_N\right)\right)\right)\left(T_x D_k \otimes T_t\right)$.

As we can see, this process is computationally cumbersome, and due to the loss of sparsity during the coefficient-to-grid conversions, it results in a dense Jacobian matrix that significantly requires more storage availability.

To address this challenge, we propose an alternative approach: approximate the iterate $u^{(k)}(x,t)$ on the left-hand side of (6.3) by its value at the final time t_M, i.e., $u^{(k)}(x,t_M)$, while leaving the iterate $u^{(k)}(x,t)$ on the right-hand side of (6.3) remains unchanged. We refer to this approach as the final-time Jacobian matrix method.

We then solve the approximate equation subject to the boundary conditions in (6.4). For every updated iterate $u^{(k)}(x,t)$, we replace it by $u^{(k)}(x,t_M)$ on the left-hand side of (6.3) and solve for $\delta u^{(k)}$ until the convergence criterion $\|\delta \mathbf{u}_k\| \leq \epsilon$ is met. In the following numerical experiments, we will see that the approximate Jacobian matrix corresponding to the final-time Jacobian matrix approach is much sparser than the full Jacobian matrix in (6.3).

The quality of the initial guess $u^{(0)}(x,t)$ can be guaranteed using the N-step spectral time-stepping approach

with Δt being sufficiently small. Specifically, when Δt is small enough, nonlinear effects remain minimal, enabling Newton's iteration to converge from the initial guess. The coefficients computed at $t = \Delta t$ subsequently serve as the initial condition for the next subproblem. This process continues until the final target time $t = t_M$.

6.2 Numerical examples

Like Section 4.4, we have in total six benchmark problems, organized into three pairs corresponding to each type of basis function. Within each pair, one problem has an explicit solution, while the other does not.

We compare the performance of the exact Jacobian matrix approach with that of the final-time Jacobian matrix method. Specifically, we will compare the number of nonzero entries in the Jacobian matrices in Newton's first iteration. For problems with explicit solutions, we will also compare the pointwise absolute errors at the final time $t = t_M$ of these two approaches. In addition, the convergence rates of the residuals are also compared. The contour plot of either the exact solution (when available) or the computed numerical solution (in the absence of an explicit solution) is presented to see the evolution of the initial condition. For problems without explicit solutions, we first compute the pointwise values at $t = t_M$ for both methods across a set of uniform grid points over the (truncated) spatial domain and then compute the absolute difference between these values. Since the spectral time-stepping approach can always be applied to problems defined over large time intervals, we focus here on shorter time intervals, specifically setting $t_0 = 0$ and $t_M = 0.1$ in the following numerical examples. However, the contour plots will show the evolutions of the initial conditions for longer periods.

6.2.1 Finite spatial domain with non-periodic boundary conditions

For our first problem with an explicit solution, we consider the classical KdV equation:

$$u_t + 6uu_x + u_{xxx} = 0, \quad (x,t) \in [-30, 30] \times [0, 0.1], \tag{6.5}$$

subject to the boundary conditions $u(-30,t) = u(30,t) = u_x(30,t) = 0$ and the initial condition $u(x,0) = \text{sech}^2\left(\frac{x-1}{\sqrt{2}}\right)$. The exact solution for this problem is explicitly known as $u(x,t) = \text{sech}^2\left(\frac{x-2t-1}{\sqrt{2}}\right)$.

Implementing Newton's iteration, we first obtain the initial guess $u^{(0)}(x,t)$ by solving the simpler linear problem

$$u_t^{(0)} + u_{xxx}^{(0)} = 0, \tag{6.6}$$

subject to the boundary conditions $u^{(0)}(-30,t) = u^{(0)}(30,t) = u_x^{(0)}(30,t) = 0$ and the initial condition

$$u^{(0)}(x,0) = \text{sech}^2\left(\frac{x-1}{\sqrt{2}}\right). \tag{6.7}$$

We then iteratively solve for the increments $\delta u^{(k)}$, $k = 0, 1, 2, \ldots$, satisfying the linearized equation

$$\delta u_t^{(k)} + \delta u_{xxx}^{(k)} + 6u^{(k)}\delta u_x^{(k)} + 6u_x^{(k)}\delta u^{(k)} = -6u^{(k)}u_x^{(k)} - u_t^{(k)} - u_{xxx}^{(k)}, \tag{6.8}$$

subject to the boundary conditions

$$\delta u^{(k)}(-30,t) = \delta u^{(k)}(30,t) = \delta u_x^{(k)}(30,t) = 0 \tag{6.9}$$

and the trivial initial condition

$$\delta u^{(k)}(x,0) = 0, \tag{6.10}$$

until the convergence criterion $\|\delta \mathbf{u}^{(k)}\| \leq 10^{-14}$ is achieved ($\epsilon = 10^{-14}$ will be used for all of the following examples). For the final-time Jacobian matrix approach, we solve instead

$$\delta u_t^{(k)} + \delta u_{xxx}^{(k)} + 6u^{(k)}(x,0.1)\delta u_x^{(k)} + 6u_x^{(k)}(x,0.1)\delta u^{(k)} = -6u^{(k)}u_x^{(k)} - u_t^{(k)} - u_{xxx}^{(k)}, \tag{6.11}$$

subject to (6.9) and (6.10). The `Julia` code used to solve (6.6) using `OperatorApproximation.jl` is as follows, where `x0` $= -30$, `x1` $= 30$, `t0` $= 0$, `t1` $= 0.1$, and `ui` contains the coefficients of the initial condition (6.7).

```
id = sparse(I, n_x, n_x); id2 = sparse(I, n_t, n_t)
Z1 = spzeros(1, n_t); Z2 = spzeros(n_t-1, n_t); Z3 = spzeros(3, n_x); Z4 = spzeros(n_x-3, n_x)
gdx = UltraMappedInterval(x0, x1, 0.0); gvx = GridValues(gdx)
spx = Ultraspherical(0, gdx); spx1 = Ultraspherical(1, gdx); spx3 = Ultraspherical(3, gdx)
C1 = Matrix(Conversion(spx3)*spx, n_x-3, n_x); C2 = Matrix(Derivative(3)*spx, n_x-3, n_x)
CC1 = vcat(Z3, C1); CC2 = vcat(Z3, C2)
Bx1 = Matrix(Conversion(FixedGridValues([x0,x1], gdx))*spx, 2, n_x)
Bx2 = Matrix(Conversion(FixedGridValues([x1], gdx))*Derivative()*spx, 1, n_x)
BBx = vcat(Bx1, Bx2, Z4); gdt = UltraMappedInterval(t0, t1, 0.0); gvt = GridValues(gdt)
spt = Ultraspherical(0, gdt); spt1 = Ultraspherical(1, gdt)
```

```
A1 = Matrix(Derivative(1)*spt, n_t-1, n_t); A2 = Matrix(Conversion(spt1)*spt, n_t-1, n_t)
AA1 = vcat(Z1, A1); AA2 = vcat(Z1, A2); u0 = BasisExpansion(x -> ui(x), spx, n_x)
Bt = Matrix(Conversion(FixedGridValues([t0], gdt)) * spt, 1, n_t); BBt = vcat(Bt, Z2)
F = zeros(n_t-1, n_x); F = vcat(transpose(ui), F)
L = kron(id, BBt) + kron(CC1, AA1) + kron(CC2, AA2) + kron(BBx, id2)
bl, bu = bandwidths(kron(CC1, AA1) + kron(CC2, AA2))
k = 3*n_t; B, Ub, V = generate_data2(copy(L), n_t*n_x, copy(vec(F)), k, bu)
XX = woodbury_sparseqr(B, Ub, V, k, bl); X = reshape(XX, n_t, n_x)
```

Before we apply either the full Jacobian matrix approach or the final-time Jacobina matrix approach, we first define the following conversion matrices between grid points and coefficients:

```
T1 = Matrix(Conversion(gvt)*spt, n_t, n_t); T4 = Matrix(Conversion(spt)*gvt, n_t, n_t)
T2 = Matrix(Conversion(gvx)*spx, n_x, n_x); T5 = Matrix(Conversion(spx)*gvx, n_x, n_x)
T3 = Matrix(Conversion(gvx)*spx1, n_x, n_x)
```

After we get the coefficient matrix of the initial guess and the above conversion matrices, we can implement the following `Julia` code following the full Jacobian matrix approach, where $\texttt{tol} = 10^{-14}$ is set for all problems.

```
C3 = Matrix(Conversion(spx3)*spx1,n_x-3,n_x); CC3 = vcat(Z3, C3)
C4 = Matrix(Derivative(1)*spx,n_x,n_x)
for j in 1:50
    O1 = kron(CC3*T5,AA2*T4)*spdiagm(vec(6*T1*X*transpose(T2)))*kron(T2*C4,T1)
    O2 = kron(CC1*T5,AA2*T4)*spdiagm(vec(6*T1*X*transpose(T3*C4)))*kron(T2,T1); L2 = L + O1 + O2
    Xn = 6*(T1*X*transpose(T2)).*(T1*X*transpose(T3*Matrix(Derivative(1)*spx,n_x,n_x)))
    Xn .= T4*Xn*transpose(T5); F = zeros(n_t, n_x)
    F[2:end,4:end] = -A2*Xn*transpose(C1) - A1*X*transpose(C1) - A2*X*transpose(C2)
    XXn = L2 \ vec(F); res = norm(XXn); XX = XX + XXn; X = reshape(XX, n_t, n_x)
    if res < tol
        break
    end
end
```

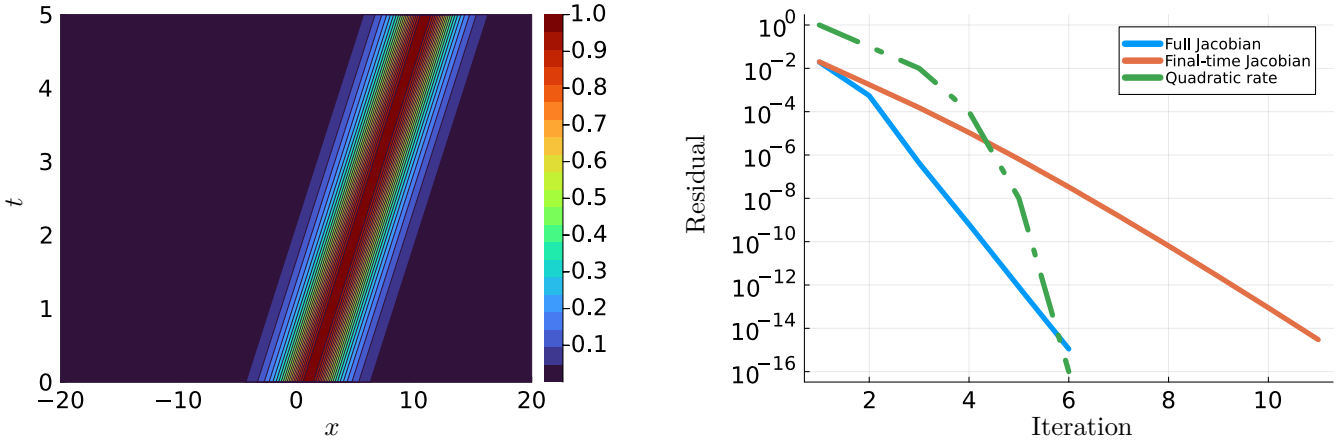
To implement the final-time Jacobian matrix approach, we first define the following function to recover the coefficients of $u_k(x,t)$ at t_M:

```
function eval2D(X, t)
    a = Matrix(Conversion(FixedGridValues([t], gdt))*spt, 1, n_t)
    return BasisExpansion(spx, (a*X)[:])
end
```

Then we implement the following Julia code for the final-time Jacobian matrix approach.

```
C4 = Matrix(Derivative(1)*spx,n_x,n_x)
for j in 1:50
    u_val = eval2D(X, t1); Mu = Multiplication(x -> 6*u_val(x)); DN = u -> Derivative()*spx*u
    ux = DN(u_val)
    Mux = Multiplication(x -> 6*ux(x))
    C3 = Matrix(Mu*Derivative(1)*spx, n_x-3, n_x); CC3 = vcat(Z3, C3)
    C4 = Matrix(Mux*spx, n_x - 3, n_x); CC4 = vcat(Z3, C4); L2 = L + kron(CC3 + CC4, AA2)
    Xn = 6*(T1*X*transpose(T2)).*(T1*X*transpose(T3*C4)); Xn .= T4*Xn*transpose(T5)
    F = zeros(n_t, n_x)
    F[2:end, 4:end] = -A2*Xn*transpose(C1) - A1*X*transpose(C1) - A2*X*transpose(C2)
    XXn = L2 \ vec(F)
    res = norm(XXn)
    XX = XX + XXn; X = reshape(XX, n_t, n_x)
    if res < tol
        break
    end
end
```

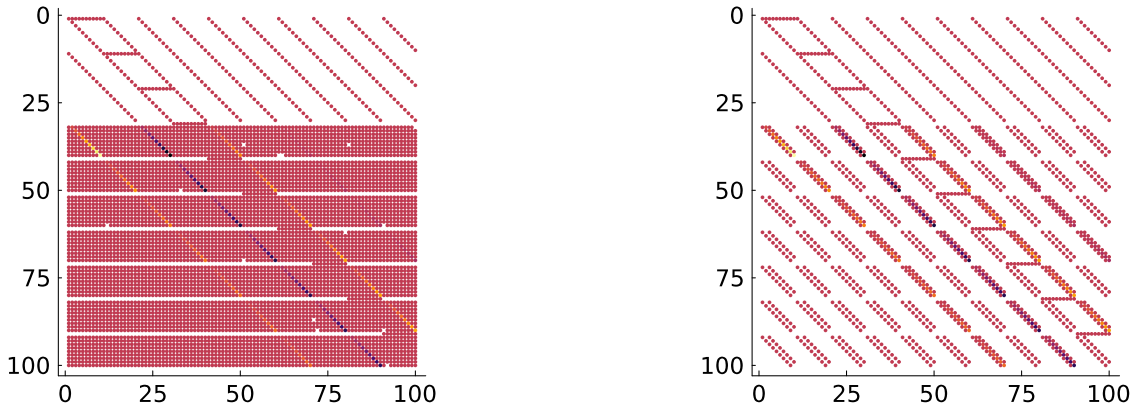
Figure 6.1(a) shows the contour plot of the exact solution from $t = 0$ to $t = 5$, which shows the propagation of the solitary traveling wave whose shape is preserved. In Figure 6.1(b), we compare the residuals of the full Jacobian matrix approach with the final-time Jacobian matrix approach, along with a reference line whose decay rate is quadratic. We can see that the residuals computed using the full Jacobian matrix exhibit approximately quadratic



(a) Contour plot of $u(x,t)$ (b) Residuals of the two approaches

Figure 6.1: (a) Contour plot of the exact solution on $(x,t) \in [-5,5] \times [0,0.1]$. (b) Semi-log plot of the residual versus iteration count comparing the full Jacobian matrix approach and the final-time Jacobian matrix approach with $n_x = 500$ and $n_t = 30$.

decay, which requires only 6 iterations to meet the convergence criterion. In contrast, residuals computed using the final-time Jacobian matrix approach decay only linearly, which requires 11 iterations to meet the convergence criterion.

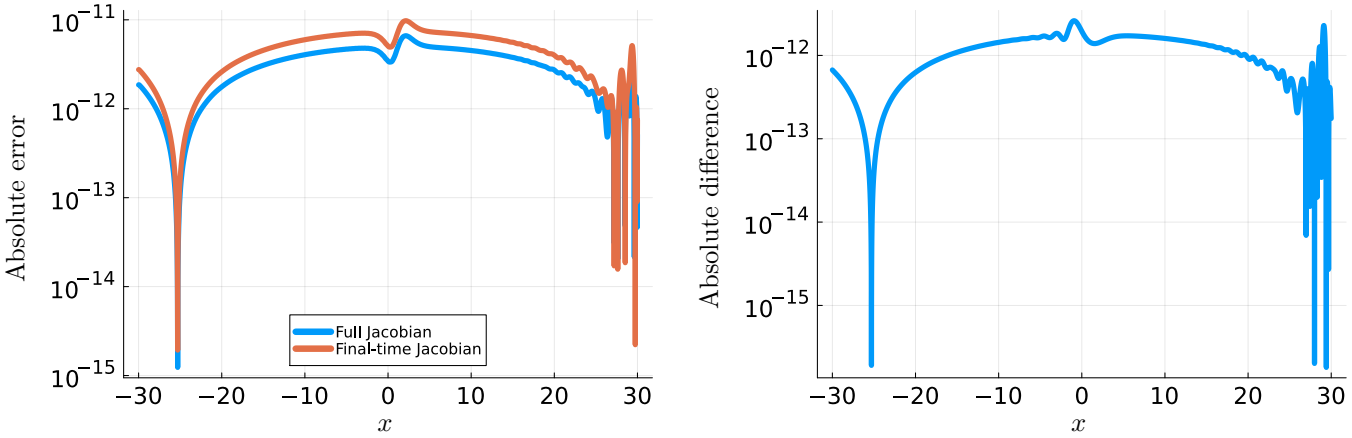


(a) The full Jacobian matrix (b) The final-time Jacobian matrix

Figure 6.2: Comparison of the Jacobian matrices in (6.8) and (6.11) with $n_x = n_t = 10$: (a) the full Jacobian matrix approach yields a dense matrix, whereas (b) the final-time Jacobian matrix approach yields a sparser matrix.

In Figure 6.2, we examine the number of nonzero entries in the Jacobian matrices of these two approaches. The matrix corresponding to the full Jacobian matrix approach is dense. In contrast, the matrix associated with the final-time Jacobian matrix approach is much sparser, which helps relieve the storage requirement at each iteration, although it comes at the cost of slower residual decay. We also find that although the Jacobian matrix in (6.11) is not almost banded, a specialized QR algorithm may be designed to exploit its special structure. In

the `Julia` code above, we use the built-in backslash solver for both approaches.



(a) Pointwise absolute errors

(b) Pointwise absolute differences

Figure 6.3: (a) Pointwise absolute errors at $t = 0.1$ computed using one global spectral-in-time solve with $n_x = 500$ and $n_t = 30$. The values are evaluated at 6001 uniform grid points between -30 and 30. (b) Pointwise absolute differences at $t = 0.1$ computed using one global spectral-in-time solve with $n_x = 500$ and $n_t = 20$. The values are evaluated at 6001 uniform grid points between -30 and 30.

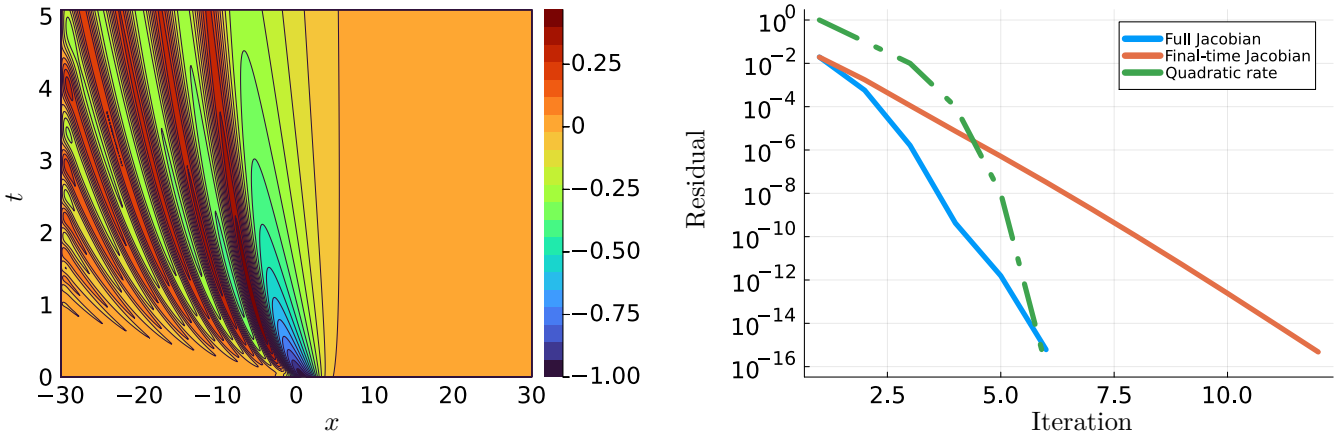
Finally, Figure 6.3(a) compares the pointwise absolute errors at $t = 0.1$ between these two approaches. We can clearly see that the final-time Jacobian matrix approach achieves comparable accuracy as the full Jacobian matrix approach, approximately $\mathcal{O}(10^{-12})$. This result shows that while the full Jacobian matrix method is advantageous due to its rapid residual convergence, the final-time Jacobian matrix approach, despite a slower rate of residual decay, can yield equally accurate results and improve the storage efficiency due to its sparser Jacobian matrix.

For our next problem without an explicit solution, we can consider the same KdV equation as before but add a minus sign to the initial condition:

$$u_t + 6uu_x + u_{xxx} = 0, \quad (x,t) \in [-30, 30] \times [0, 0.1], \tag{6.12}$$

subject to the boundary conditions $u(-30,t) = u(30,t) = u_x(30,t) = 0$ and the initial condition $u(x,0) = -\text{sech}^2\left(\frac{x-1}{\sqrt{2}}\right)$. The `Julia` code used to solve this problem is the same as before except that now we have added a minus sign to the initial condition.

Figure 6.4(a) shows the contour plot of the numerical solution from $t = 0$ to $t = 5$, showing the evolution of the initial condition under the nonlinear dynamics characteristic of the KdV equation. In Figure 6.4(b), we compare the residuals of the full Jacobian matrix approach with the final-time Jacobian matrix approach, alongside with a reference line with a linear rate of decay. We can see that with $\epsilon = 10^{-14}$, the residuals of the final-time



(a) Contour plot of the numerical solution

(b) Residuals of the two approaches

Figure 6.4: (a) Contour plot of the numerical solution on $(x,t) \in [-30,30] \times [0,5]$ using a 50-step time-stepping approach with $n_x = 500$ and $n_t = 20$. (b) Semi-log plot of the residual versus iteration count comparing the full Jacobian matrix approach and the final-time Jacobian matrix approach with $n_x = 120$ and $n_t = 30$.

Jacobian matrix approach have a linear decay, whereas the residuals the full Jacobian matrix approach have a quadratic decay.

Since the new initial condition does not affect the Jacobian matrix using either approach, we still have a sparser Jacobian matrix using the final-time Jacobian matrix approach than using the full Jacobian matrix approach.

Finally, Figure 6.3(b) shows the pointwise absolute difference between these two methods at $t = 0.1$. We can clearly see that the difference between them remains in $\mathcal{O}(10^{-12})$, which shows that the final-time Jacobian matrix approach can produce results that are comparable to the full Jacobian matrix approach.

6.2.2 Finite spatial domain with periodic boundary conditions

For our first problem with an explicit solution, we consider the following equation:

$$u_t + uu_x - u_{xx} = -\sin(t)\sin(x) + \cos^2(t)\sin(x)\cos(x) + \cos(t)\sin(x), \quad (x,t) \in [0,2\pi] \times [0,0.1], \tag{6.13}$$

subject to the periodic boundary conditions

$$u(0,t) = u(2\pi,t), \quad u_x(0,t) = u_x(2\pi,t), \tag{6.14}$$

and the initial condition

$$u(x,0) = \sin(x). \tag{6.15}$$

The exact solution to this problem is $u(x,t) = \cos(t)\sin(x)$.

Implementing Newton's iteration, we first obtain the initial guess $u^{(0)}(x,t)$ by solving the simpler forced heat equation:

$$u_t^{(0)} - u_{xx}^{(0)} = -\sin(t)\sin(x) + \cos^2(t)\sin(x)\cos(x) + \cos(t)\sin(x), \tag{6.16}$$

subject to the periodic boundary conditions

$$u^{(0)}(0,t) = u^{(0)}(2\pi,t), \quad u_x^{(0)}(0,t) = u_x^{(0)}(2\pi,t), \tag{6.17}$$

and the initial condition

$$u^{(0)}(x,0) = \sin(x). \tag{6.18}$$

We then iteratively solve for the increment $\delta u^{(k)}$, $k = 0, 1, 2, \ldots$, satisfying the linearized equation:

$$\delta u_t^{(k)} - \delta u_{xx}^{(k)} + u^{(k)}\delta u_x^{(k)} + u_x^{(k)}\delta u^{(k)} = -\sin(t)\sin(x) + \cos^2(t)\sin(x)\cos(x) + \cos(t)\sin(x) - u^{(k)}u_x^{(k)} - u_t^{(k)} + u_{xx}^{(k)}, \tag{6.19}$$

subject to the periodic boundary conditions

$$\delta u_k(0,t) = \delta u_k(2\pi,t), \quad \delta u_{kx}(0,t) = \delta u_{kx}(2\pi,t), \tag{6.20}$$

and the trivial initial condition (6.10). For the final-time Jacobian matrix approach, we replace $u^{(k)}$ and $u_x^{(k)}$ on the left-hand side of (6.19) by $u^{(k)}(x,0.1)$ and $u_x^{(k)}(x,0.1)$, respectively.

The `Julia` code used to solve (6.16) using `operatorApproximation.jl` is as follows, where `t0` $= 0$, `t1` $= 0.1$, and the initial condition `ui` is (6.18).

```
id = sparse(I, n_x, n_x); Z1 = spzeros(1, n_t); Z2 = spzeros(n_t - 1, n_t)
gdf = PeriodicMappedInterval(0, 2 π); spf = Fourier(gdf); gvx = GridValues(gdf)
C1 = id; C2 = -Matrix(Derivative(2)*spf, n_x, n_x)
gdt = UltraMappedInterval(t0, t1, 0.0); gvt = GridValues(gdt)
spt = Ultraspherical(0, gdt); spt1 = Ultraspherical(1, gdt)
A1 = Matrix(Derivative(1)*spt, n_t - 1, n_t); A2 = Matrix(Conversion(spt1)*spt, n_t - 1, n_t)
AA1 = vcat(Z1, A1); AA2 = vcat(Z1, A2)
Bt = reshape([(-1)^(j+1) * sqrt(2) for j in 1:n_t], 1, n_t); Bt[1] = 1; BBt = vcat(Bt, Z2)
c_n = BasisExpansion(t -> -sin(t), spt1, n_t-1).c; c_k = BasisExpansion(x -> sin(x), spf, n_x).c
```

```
c_n1 = BasisExpansion(t -> (cos(t))^2, spt1, n_t-1).c
c_k1 = BasisExpansion(x -> sin(x)*cos(x), spf, n_x).c
c_n2 = BasisExpansion(t -> cos(t), spt1, n_t-1).c; c_k2 = BasisExpansion(x -> sin(x), spf, n_x).c
F = c_n * transpose(c_k) + c_n1 * transpose(c_k1) + c_n2 * transpose(c_k2)
F = vcat(transpose(ui),F)
L = kron(id, BBt) + kron(C1, AA1) + kron(C2, AA2)
bl, bu = bandwidths(L); XX = sparse_qr(copy(L), vec(F), bl); X = reshape(XX, n_t, n_x)
```

Before implementing either approach, we first define the following conversion operators.

```
T1 = Matrix(Conversion(gvt)*spt, n_t, n_t); T3 = Matrix(Conversion(spt)*gvt, n_t, n_t)
T2 = Matrix(Conversion(gvx)*spf, n_x, n_x); T4 = Matrix(Conversion(spf)*gvx, n_x, n_x)
```

Then the full Jacobian matrix approach can be implemented using the following code.

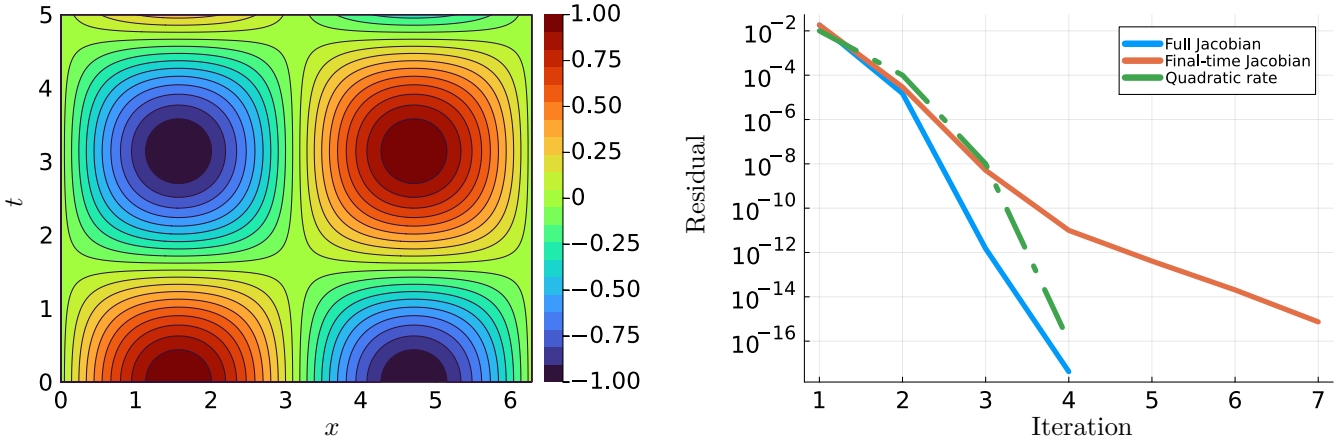
```
C4 = Matrix(Derivative(1)*spf, n_x, n_x)
for j in 1:50
    O1 = kron(T4, AA2*T3)*spdiagm(vec(T1*X*transpose(T2)))*kron(T2*C4,T1)
    O2 = kron(T4, AA2*T3)*spdiagm(vec(T1*X*transpose(C4)*transpose(T2)))*kron(T2,T1)
    L2 = L + O1 + O2
    Xn = (T1*X*transpose(T2)).*(T1*X*transpose(T2*C4)); Xn .= T3*Xn*transpose(T4)
    F = zeros(ComplexF64,n_t,n_x)
    F[2:end,1:end] = c_n * transpose(c_k) + c_n1 * transpose(c_k1) + c_n2 * transpose(c_k2)
    - A2*Xn - A1*X - A2*X*transpose(C2)
    XXn = L2 \ vec(F); res = norm(XXn)
    XX = XX + XXn; X = reshape(XX, n_t, n_x)
    if res < tol
        break
    end
end
```

Like before, we again first define the following function to recover the coefficients of $u^{(k)}(x,t)$ at t_M:

```
function eval2D(X, t)
    a = Matrix(Conversion(FixedGridValues([t], gdt)) * spt, 1, n_t)
    return BasisExpansion(spf, (a * X)[:])
end
```

The final-time Jacobian matrix approach can then be implemented using the following codes.

```
for j in 1:50
    u_val = eval2D(X, t1)
    Mu = Multiplication(x -> u_val(x))
    DN = u -> Derivative() * spf * u
    ux = DN(u_val)
    Mux = Multiplication(x -> ux(x))
    C3 = Matrix(Mu*Derivative(1)*spf,n_x,n_x); C4 = Matrix(Mux*spf,n_x,n_x)
    L2 = L + kron(C3 + C4, AA2)
    Xn = (T1*X*transpose(T2)).*(T1*X*transpose(T2*Matrix(Derivative(1)*spf,n_x,n_x)))
    Xn .= T3*Xn*transpose(T4)
    F = zeros(ComplexF64,n_t,n_x)
    F[2:end,1:end] = c_n * transpose(c_k) + c_n1 * transpose(c_k1) + c_n2 * transpose(c_k2)
    - A2*Xn - A1*X - A2*X*transpose(C2)
    bl, bu = bandwidths(L2); XX = sparse_qr(L2, vec(F), bl)
    res = norm(XXn); XX = XX + XXn; X = reshape(XX, n_t, n_x)
    if res < tol
        break
    end
end
```

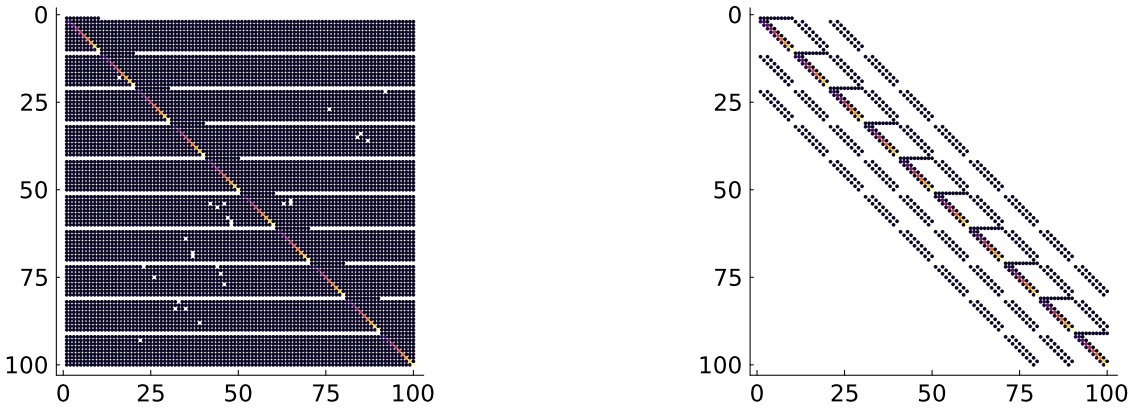


(a) Contour plot of $u(x,t)$

(b) Residuals of the two approaches

Figure 6.5: (a) Contour plot of the exact solution on $(x,t) \in [0,2\pi] \times [0,5]$. (b) Semi-log plot of the residual versus iteration count comparing the full Jacobian matrix approach and the final-time Jacobian matrix approach with $n_x = 16$ and $n_t = 8$.

Figure 6.5(a) shows the contour plot of the exact solution on the periodic spatial domain $[0, 2\pi]$ from $t = 0$ to $t = 5$. In Figure 6.5(b), we compare the residuals of the full Jacobian matrix approach with the final-time Jacobian matrix approach, along with a reference line with a quadratic decay rate. We can see that the residuals computed using the full Jacobian matrix exhibit quadratic convergence and meet the convergence criterion in just 4 iterations. In contrast, residuals using the final-time Jacobian matrix approach decay only linearly, requiring 7 iterations to meet the convergence criterion.



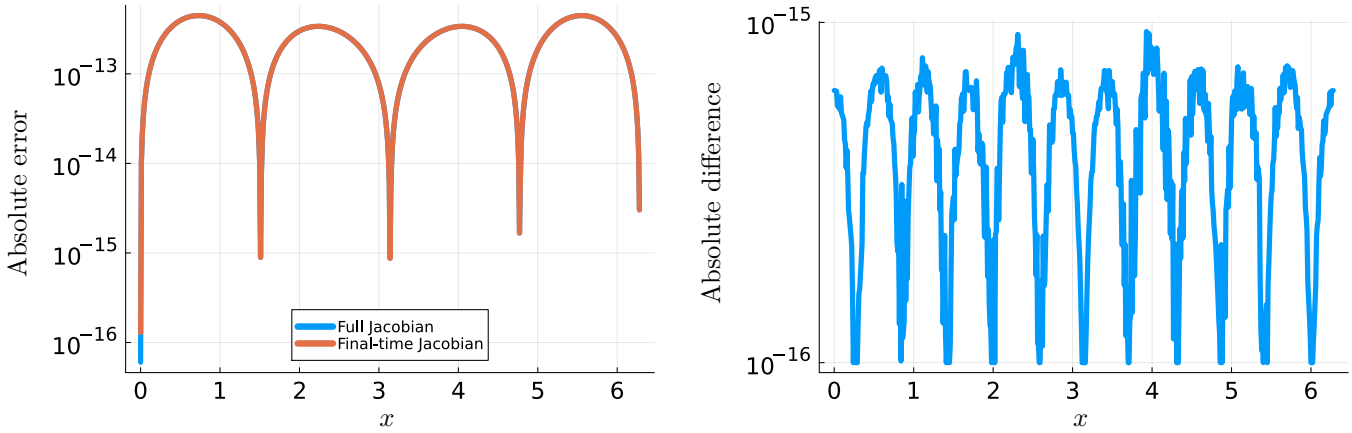
(a) The full Jacobian matrix

(b) The final-time Jacobian matrix

Figure 6.6: Comparison of the Jacobian matrices with $n_x = n_t = 10$: (a) the full Jacobian matrix approach yields a dense matrix, whereas (b) the final-time Jacobian matrix approach yields a sparser and banded matrix.

In Figure 6.6, we examine the number of nonzero entries in the Jacobian matrices of these two approaches. The Jacobian matrix corresponding to the correct approach is dense. In contrast, the Jacobian matrix associated

with the other approach is much sparser and banded, which is expected since multiplication matrices in Fourier space are at worst banded. This is encouraging since we can apply Algorithm 5 to solve it in each iteration.



(a) Pointwise absolute errors

(b) Pointwise absolute difference

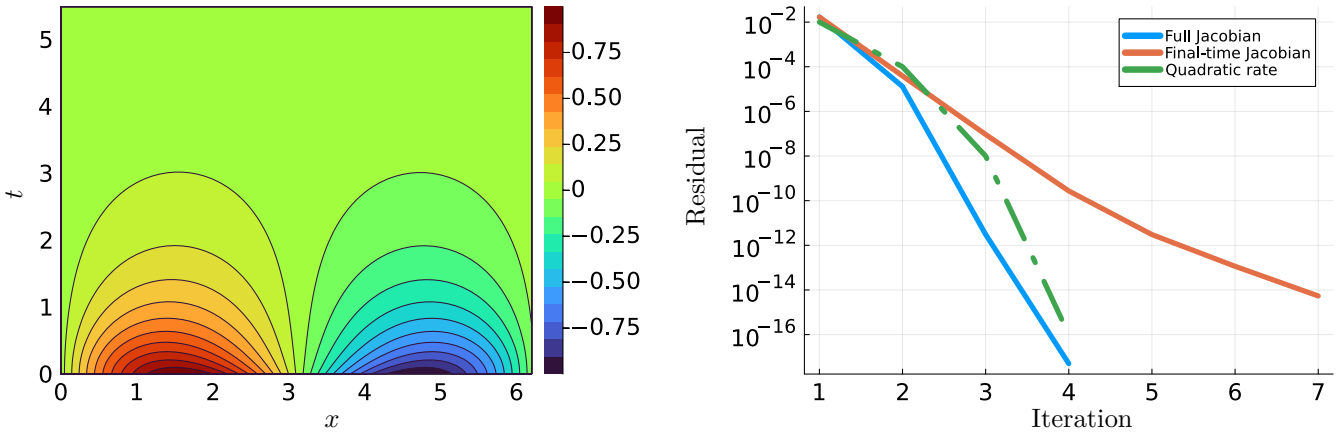
Figure 6.7: (a) Pointwise absolute errors at $t = 0.1$ computed using one global spectral-in-time solve with $n_x = 16$ and $n_t = 6$. The values are evaluated at 629 uniform grid points between 0 and 2π. (b) Pointwise absolute differences at $t = 0.1$ computed using one global spectral-in-time solve with $n_x = 12$ and $n_t = 4$. The values are evaluated at 629 uniform grid points between 0 and 2π.

Finally, Figure 6.7(a) compares the pointwise absolute errors at $t = 0.1$ between these two approaches. We can clearly see that both achieve $\mathcal{O}(10^{-13})$, which shows that the final-time Jacobian matrix approach should be used because of the resulting sparse and banded Jacobian matrix.

For our next problem without an explicit solution, we can consider the previous one without the forcing term:

$$u_t + uu_x - u_{xx} = 0, \quad (x,t) \in [0, 2\pi] \times [0, 0.1], \tag{6.21}$$

subject to the periodic boundary conditions (6.14) and the same initial condition (6.15).



(a) Contour plot of the numerical solution

(b) Residuals of the two approaches

Figure 6.8: (a) Contour plot of the numerical solution on $(x,t) \in [0, 2\pi] \times [0, 5]$ using a 10-step time-stepping approach with $n_x = 21$ and $n_t = 20$. (b) Semi-log plot of the residual versus iteration count comparing the full Jacobian matrix approach and the final-time Jacobian matrix approach with $n_x = 16$ and $n_t = 8$.

Figure 6.8(a) shows the contour plot of the numerical solution on the periodic spatial domain $[0, 2\pi]$ from $t = 0$ to $t = 5$. In Figure 6.8(b), we compare the residuals of the full Jacobian matrix approach with the final-time Jacobian matrix approach, alongside with a reference line with a quadratic rate of decay. We can again see that the residuals computed using the full Jacobian matrix exhibit quadratic convergence and meet the convergence criterion in just 4 iterations. Although residuals using the final-time Jacobian matrix approach decay linearly, it still requires only 7 iterations to meet the convergence criterion.

Since removing the forcing term does not change the Jacobian matrix using either approach, we still have a sparser and banded Jacobian matrix using the final-time Jacobian matrix approach than using the full Jacobian matrix approach.

Finally, Figure 6.7(b) shows the pointwise absolute differences between these two methods at $t = 0.1$. We can clearly see that the difference between them achieves machine precision. Based on these two problems, we can conclude that the final-time Jacobian matrix approach should be used to numerically solve nonlinear time-evolution PDEs with periodic boundary conditions.

6.2.3 Unbounded spatial domain

For our first problem with an explicit solution, we consider the KdV equation:

$$u_t + 6uu_x + u_{xxx} = 0, \quad (x,t) \in \mathbb{R} \times [0, 0.1], \tag{6.22}$$

subject to the boundary conditions

$$u(x,t) \to 0 \text{ as } x \to \pm\infty \tag{6.23}$$

and the initial condition $u(x,0) = \text{sech}^2\left(\frac{x-1}{\sqrt{2}}\right)$. The exact solution for this problem is known explicitly as $u(x,t) = \text{sech}^2\left(\frac{x-2t-1}{\sqrt{2}}\right)$. The `Julia` code used to solve for the initial guess is as follows.

```
id = sparse(I, n_x, n_x); id2 = sparse(I, n_t, n_t)
Z1 = spzeros(1, n_t); Z2 = spzeros(n_t-1, n_t); Z3 = spzeros(1, n_x); Z4 = spzeros(n_x-1, n_x)
gdx = RationalRealAxis(); gvx = GridValues(gdx); spx = OscRational(gdx,0.0)
C1 = sparse(I,n_x,n_x)[2:end,1:end]; C2 = Matrix(Derivative(3)*spx,n_x-1,n_x)
CC1 = vcat(Z3, C1); CC2 = vcat(Z3, C2)
Bx = ones(1,n_x); BBx = vcat(Bx, Z4)
gdt = UltraMappedInterval(t0, t1, 0.0); gvt = GridValues(gdt)
spt = Ultraspherical(0, gdt); spt1 = Ultraspherical(1, gdt)
A1 = Matrix(Derivative(1)*spt,n_t-1,n_t); A2 = Matrix(Conversion(spt1)*spt,n_t-1,n_t)
AA1 = vcat(Z1, A1); AA2 = vcat(Z1, A2)
Bt = reshape([(-1)^(j+1) * sqrt(2) for j in 1:n_t],1,n_t); Bt[1] = 1; BBt = vcat(Bt, Z2)
F = zeros(n_t-1, n_x); F = vcat(transpose(ui), F)
L = kron(id, BBt) + kron(CC1, AA1) + kron(CC2, AA2) + kron(BBx, id2)
bl, bu = bandwidths(kron(CC1, AA1) + kron(CC2, AA2))
k = n_t; B3,Ub,V = generate_data2(copy(L), n_t*n_x, copy(vec(F)), k, bu)
XX = woodbury_sparseqr(B3, Ub, V, k, bl); X = reshape(XX, n_t, n_x)
```

Before implementing either approach, we again first define the following conversion operators.

```
T1 = Matrix(Conversion(gvt)*spt, n_t, n_t); T3 = Matrix(Conversion(spt)*gvt, n_t, n_t)
T2 = Matrix(Conversion(gvx)*spx, n_x, n_x); T4 = Matrix(Conversion(spx)*gvx, n_x, n_x)
```

Then, we implement the following code for the full Jacobian matrix approach.

```
C4 = Matrix(Derivative(1)*spx,n_x,n_x)
for j in 1:50
```

```
    O1 = kron(CC1*T4,AA2*T3)*spdiagm(vec(6*T1*X*transpose(T2)))*kron(T2*C4,T1)
    O2 = kron(CC1*T4,AA2*T3)*spdiagm(vec(6*T1*X*transpose(T2*C4)))*kron(T2,T1)
    L2 = L + O1 + O2
    Xn = 6*(T1*X*transpose(T2)).*(T1*X*transpose(T2*Matrix(Derivative(1)*spx,n_x,n_x)))
    Xn .= T3*Xn*transpose(T4)
    F = zeros(ComplexF64,n_t,n_x)
    F[2:end,2:end] = -A2*Xn*transpose(C1) - A1*X*transpose(C1) - A2*X*transpose(C2)
    XXn = L2 \ vec(F); res = norm(XXn); push!(res_list, res)
    XX = XX + XXn; X = reshape(XX, n_t, n_x)
    if res < tol
        break
    end
end
```

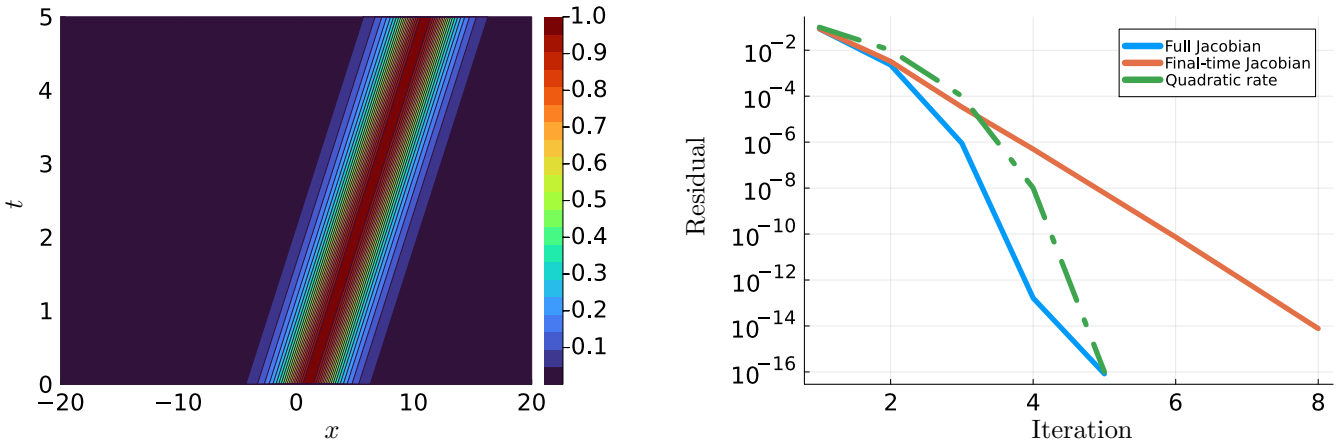
For the final-time Jacobian matrix approach, we still first define the following function to recover the coefficients at the final time Δt:

```
function eval2D(X, t)
    a = Matrix(Conversion(FixedGridValues([t], gdt))*spt, 1, n_t)
    return BasisExpansion(spx, (a*X)[:])
end
```

Then we implement the following code for the final-time Jacobian matrix approach.

```
C5 = Matrix(Derivative(1)*spx, n_x, n_x)
for j in 1:50
    u_val = eval2D(X, t1)
    Mu = Multiplication(x -> u_val(x))
    DN = u -> Derivative()*spx*u
    ux = DN(u_val)
    Mux = Multiplication(x -> ux(x))
    C3 = Matrix(Mu * Derivative(1) * spx, n_x-1, n_x); C4 = Matrix(Mux * spx, n_x-1, n_x)
    CC3 = vcat(Z3, C3); CC4 = vcat(Z3, C4)
```

```
    L2 = L + kron(CC3 + CC4, AA2)
    Xn = (T1 * X * transpose(T2)) .* (T1 * X * transpose(T2 * C5))
    Xn .= T3 * Xn * transpose(T4)
    F = zeros(ComplexF64,n_t,n_x)
    F[2:end, 2:end] = -A2 * Xn * transpose(C1) - A1 * X * transpose(C1) + A2 * X * transpose(C2)
    bl, bu = bandwidths(kron(CC3 + CC4, AA2))
    B3,Ub,V = generate_data2(L2, n_t*n_x, vec(F), k, bu)
    XXn = woodbury_sparseqr(B3, Ub, V, k, bl)
    res = norm(XXn)
    XX = XX + XXn
    X = reshape(XX, n_t, n_x)
    if res < tol
        break
    end
end
```



(a) Contour plot of $u(x,t)$

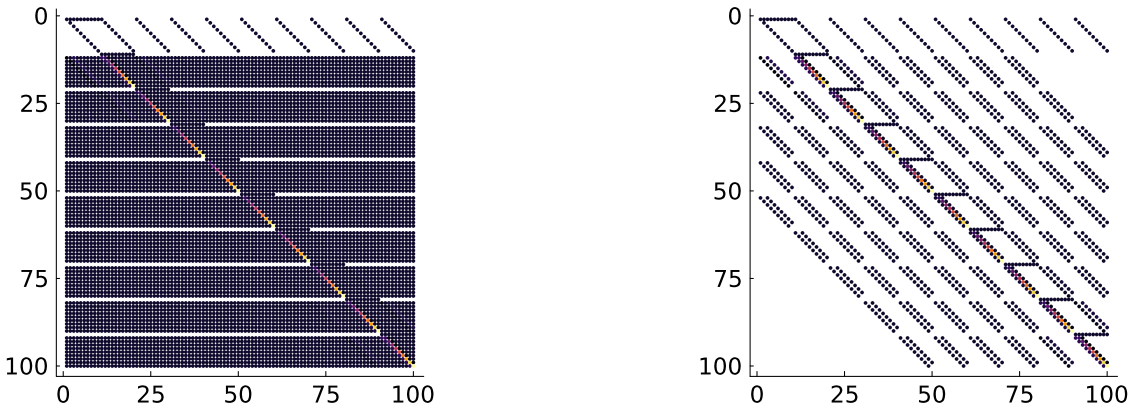
(b) Residuals of the two approaches

Figure 6.9: (a) Contour plot of the exact solution on $(x,t) \in [-20,20] \times [0,5]$. (b) Semi-log plot of the residual versus iteration count comparing the full Jacobian matrix approach and the final-time Jacobian matrix approach using $n_x = 500$ and $n_t = 30$.

Figure 6.9(a) shows the contour plot of the exact solution over the truncated spatial domain $[-20,20]$ from $t = 0$ to $t = 5$. In Figure 6.9(b), we compare the residuals of the full Jacobian matrix approach and the final-time Jacobian matrix approach, along with a reference line with a quadratic decay rate. We can see that the residuals computed using the full Jacobian matrix approach exhibit a quadratic convergence, which requires only 5 iterations, while the residual decay of the final-time Jacobian matrix approach is linear and requires 8 iterations.

In Figure 6.10, we compare the Jacobian matrices of these two approaches, showing that the matrix using the full Jacobain matrix is dense, whereas the matrix derived using the final-time Jacobian matrix approach is sparser.

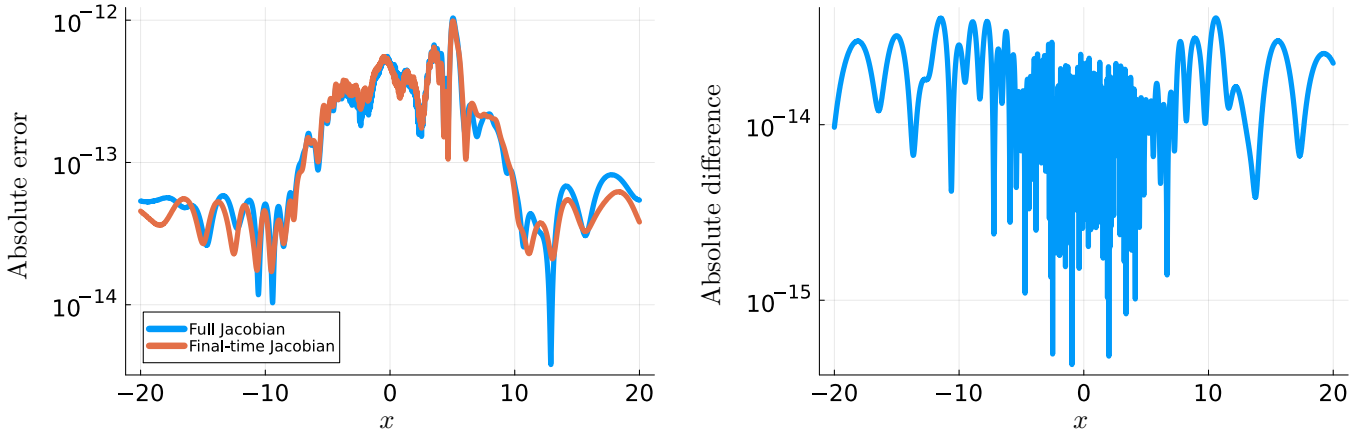
Regarding the accuracy at $t = 0.1$, Figure 6.11(a) shows the pointwise absolute errors at $t = 0.1$ for both approaches, evaluated at 4001 uniform grid points between -20 and 20. Both methods achieve essentially identical accuracy at $\mathcal{O}(10^{-13})$. Thus, the final-time Jacobian matrix approach effectively balances matrix sparsity and numerical accuracy, despite the slower residual convergence rate.



(a) The full Jacobian matrix

(b) The final-time Jacobian matrix

Figure 6.10: Comparison of the Jacobian matrices with $n_x = n_t = 10$: (a) the full Jacobian matrix approach yields a dense matrix, whereas (b) the final-time Jacobian matrix approach yields a sparser matrix.



(a) Pointwise absolute errors

(b) Pointwise absolute differences

Figure 6.11: (a) Pointwise absolute errors at $t = 0.1$ computed using one global spectral-in-time solve with $n_x = 500$ and $n_t = 30$. The values are evaluated at 4001 uniform grid points between -20 and 20. (b) Pointwise absolute differences at $t = 0.1$ computed using one global spectral-in-time solve with $n_x = 400$ and $n_t = 20$. The values are evaluated at 4001 uniform grid points between -20 and 20.

For our next problem without an explicit solution, we consider the following Kuramoto–Sivashinsky equation

[Siv77, Kur78, Siv80]:

$$u_t + u_{xx} + u_{xxxx} + uu_x = 0, \quad (x,t) \in \mathbb{R} \times [0, 0.1], \tag{6.24}$$

subject to the boundary conditions in (6.23) and the initial condition $u(x,0) = e^{-x^2}$. In implementing Newton's iteration, we begin with an initial guess $u^{(0)}(x,t)$, obtained by solving the linear equation

$$u_t^{(0)} + u_{xx}^{(0)} + u_{xxxx}^{(0)} = 0 \tag{6.25}$$

with the boundary conditions (6.23) and the initial condition $u^{(0)}(x,0) = e^{-x^2}$. We then iteratively solve for the increment $\delta u^{(k)}$, for $k = 0, 1, 2, \ldots$, satisfying the linearized equation:

$$\delta u_t^{(k)} + \delta u_{xx}^{(k)} + \delta u_{xxxx}^{(k)} + u^{(k)}\delta u_x^{(k)} + u_x^{(k)}\delta u^{(k)} = -u^{(k)}u_x^{(k)} - u_t^{(k)} - u_{xx}^{(k)} - u_{xxxx}^{(k)} \tag{6.26}$$

subject to the boundary conditions (6.23) and the trivial initial condition (6.10). For the final-time Jacobian matrix approach, we solve instead

$$\delta u_t^{(k)} + \delta u_{xx}^{(k)} + \delta u_{xxxx}^{(k)} + u^{(k)}(x,0.1)\delta u_x^{(k)} + u_x^{(k)}(x,0.1)\delta u^{(k)} = -u^{(k)}u_x^{(k)} - u_t^{(k)} - u_{xx}^{(k)} - u_{xxxx}^{(k)} \tag{6.27}$$

subject to (6.23) and (6.10). The `Julia` code used to solve (6.25) is as follows.

```
id = sparse(I, n_x, n_x); id2 = sparse(I, n_t, n_t)
Z1 = spzeros(1, n_t); Z2 = spzeros(n_t-1, n_t); Z3 = spzeros(1, n_x); Z4 = spzeros(n_x-1, n_x)
gdx = RationalRealAxis(); gvx = GridValues(gdx); spx = OscRational(gdx,0.0)
C1 = sparse(I,n_x,n_x)[2:end,1:end]; C2 = Matrix(Derivative(2)*spx,n_x-1,n_x)
C3 = Matrix(Derivative(4)*spx,n_x-1,n_x)
CC1 = vcat(Z3, C1); CC2 = vcat(Z3, C2); CC3 = vcat(Z3, C3)
Bx = ones(1,n_x); BBx = vcat(Bx, Z4)
gdt = UltraMappedInterval(t0, t1, 0.0); gvt = GridValues(gdt); spt = Ultraspherical(0, gdt)
spt1 = Ultraspherical(1, gdt)
A1 = Matrix(Derivative(1)*spt,n_t-1,n_t)
A2 = Matrix(Conversion(spt1)*spt,n_t-1,n_t)
A3 = Matrix(Conversion(spt1)*spt,n_t-1,n_t)
AA1 = vcat(Z1, A1); AA2 = vcat(Z1, A2); AA3 = vcat(Z1, A3)
u0 = BasisExpansion(x -> ui(x), spx, n_x)
```

```
Bt = Matrix(Conversion(FixedGridValues([t0], gdt)) * spt, 1, n_t); BBt = vcat(Bt, Z2)
F = zeros(n_t-1, n_x); F = vcat(transpose(ui), F)
L = kron(id, BBt) + kron(CC1, AA1) + kron(CC2, AA2) + kron(CC3, AA3) + kron(BBx, id2)
bl, bu = bandwidths(kron(CC1, AA1) + kron(CC2, AA2))
k = findfirst(i -> all(iszero, kron(BBx,id2)[i, :]), 1:size(kron(BBx,id2),1)) - 1
B3,Ub,V = generate_data2(copy(L), n_t*n_x, copy(vec(F)), k, bu)
XX = woodbury_sparseqr(B3, Ub, V, k, bl); X = reshape(XX, n_t, n_x)
```

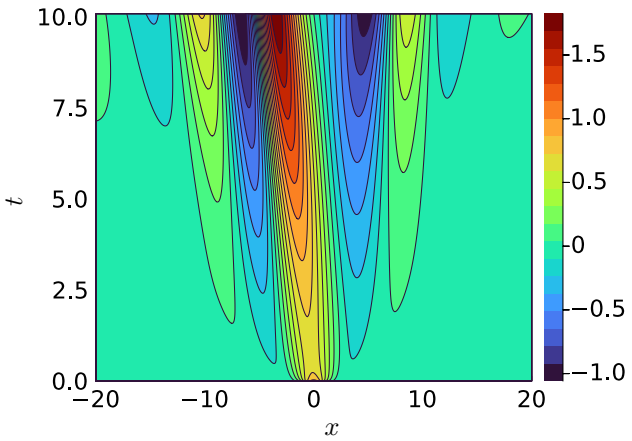
Using the same conversion operators as before, we implement the following codes for the full Jacobian matrix approach.

```
C4 = Matrix(Derivative(1)*spx, n_x, n_x)
for j in 1:50
    O1 = kron(CC1*T4,AA2*T3)*spdiagm(vec(T1*X*transpose(T2)))*kron(T2*C4, T1)
    O2 = kron(CC1*T4,AA2*T3)*spdiagm(vec(T1*X*transpose(T2*C4)))*kron(T2, T1); L2 = L + O1 + O2
    Xn = (T1*X*transpose(T2)).*(T1*X*transpose(T2*C4)); Xn .= T3*Xn*transpose(T4)
    F = spzeros(ComplexF64,n_t,n_x)
    F[2:end, 2:end] = -A2 * Xn * transpose(C1) - A1 * X * transpose(C1) - A2 * X * transpose(C2)
    - A3 * X * transpose(C3)
    XXn = L2 \ vec(F); res = norm(XXn); XX = XX + XXn; X = reshape(XX, n_t, n_x)
    if res < tol
        break
    end
end
```

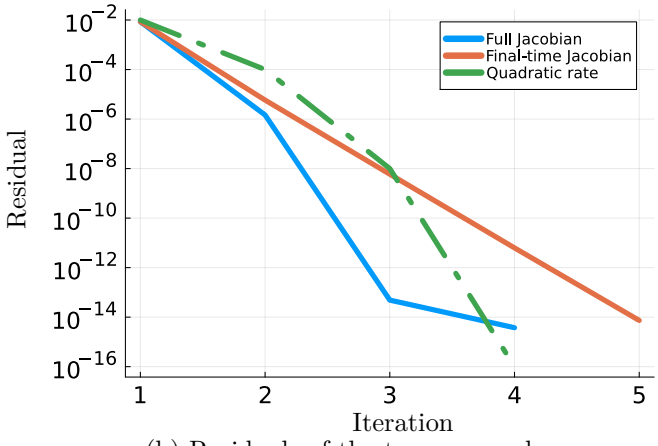
The following code is used to implement the final-time Jacobian matrix approach.

```
C6 = Matrix(Derivative(1)*spx, n_x, n_x)
for j in 1:50
    u_val = eval2D(X, t1)
    Mu = Multiplication(x -> u_val(x))
    DN = u -> Derivative() * spx * u
    ux = DN(u_val)
```

```
    Mux = Multiplication(x -> ux(x))
    C4 = Matrix(Mu * Derivative(1) * spx, n_x-1, n_x)
    C5 = Matrix(Mux * spx, n_x-1, n_x)
    CC4 = vcat(Z3, C4)
    CC5 = vcat(Z3, C5)
    L2 = L + kron(CC5 + CC4, AA2)
    Xn = (T1 * X * transpose(T2)) .* (T1 * X * transpose(T2*C6))
    Xn .= T3 * Xn * transpose(T4)
    F = zeros(ComplexF64,n_t,n_x)
    F[2:end, 2:end] = -A2 * Xn * transpose(C1) - A1 * X * transpose(C1) - A2 * X * transpose(C2)
    - A3 * X * transpose(C3)
    bl, bu = bandwidths(kron(CC5 + CC4, AA2))
    B3,Ub,V = generate_data2(L2, n_t*n_x, vec(F), k, bu)
    XXn = woodbury_sparseqr(B3, Ub, V, k, bl); res = norm(XXn); XX = XX + XXn
    X = reshape(XX, n_t, n_x)
    if res < tol
        break
    end
end
```



(a) Contour plot of the numerical solution

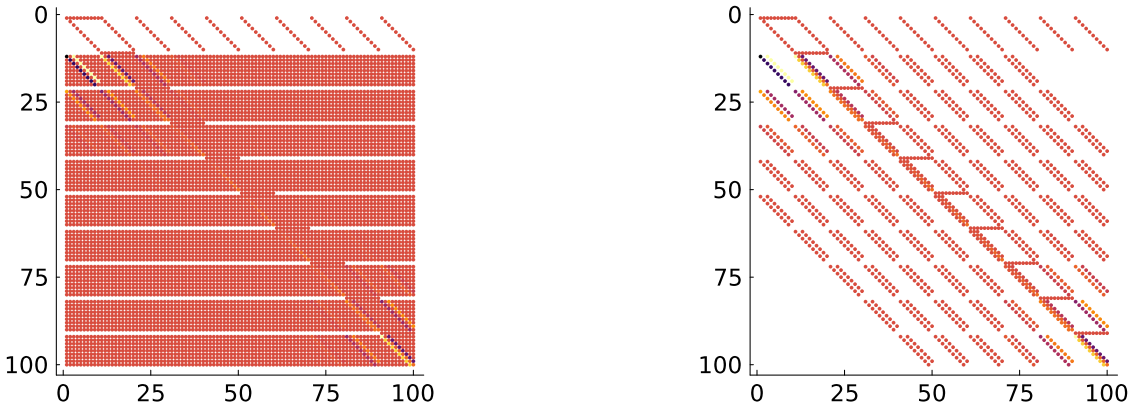


(b) Residuals of the two approaches

Figure 6.12: (a) Contour plot of the numerical solution on $(x,t) \in [-20,20] \times [0,5]$ using a 50-step time-stepping approach with $n_x = 200$ and $n_t = 50$. (b) Semi-log plot of the residual versus iteration count comparing the full Jacobian matrix approach and the final-time Jacobian matrix approach using $n_x = 400$ and $n_t = 20$.

Figure 6.12(a) shows the contour plot of the numerical solution on the truncated spatial domain $[-20,20]$

from $t = 0$ to $t = 10$, computed using a 100-step time-stepping approach with $n_x = 200$ and $n_t = 50$. We can see that the chaotic behavior of the Kuramoto-Sivashinsky equation starts to kick in after $t = 7$. In Figure 6.12(b), we compare the residuals of these two approaches, alongside with a reference line decaying quadratically. The residuals corresponding to the full Jacobian matrix approach exhibit a quadratic convergence, rapidly reaching the prescribed tolerance within just 4 iterations. The residuals for the final-time Jacobian matrix approach exhibit linear convergence, but it still only requires 5 iterations to achieve the same accuracy level.



(a) The full Jacobian matrix

(b) The final-time Jacobian matrix

Figure 6.13: Comparison of the Jacobian matrices in (6.26) and (6.27) with $n_x = n_t = 10$: (a) the full Jacobian matrix approach yields a dense matrix, whereas (b) the final-time Jacobian matrix approach yields a sparser matrix.

In Figure 6.13, we examine the number of nonzero entries of the Jacobian matrices using these two approaches, both generated with $n_x = n_t = 10$. The full Jacobian matrix approach yields a dense matrix, which causes higher storage requirements. Conversely, the final-time Jacobian matrix approach results in a sparser matrix.

Regarding the accuracy of these two approaches at $t = 0.1$, Figure 6.11(b) shows the pointwise absolute differences at $t = 0.1$, computed using 4001 spatial grid points uniformly distributed over the interval $[-20, 20]$. We can see that the differences remain on $\mathcal{O}(10^{-14})$, showing that the final-time Jacobian matrix approach achieves an accuracy essentially indistinguishable from the full Jacobian matrix approach.

Appendix A

A tutorial of the Julia commands of OperatorApproximation.jl and ApproxFun.jl

In this appendix, we will give a brief tutorial of the Julia commands used in this thesis from the two main packages: OperatorApproximation.jl [TL24] and ApproxFun.jl [OGS+15]. The first package is the one used mainly, and the second one is used when BigFloat arithmetic is used.

Using OperatorApproximation.jl, a concrete operator is constructed by multiplying a predefined abstract operator by a certain basis and a truncated matrix is obtained by truncating the concrete operator. The following list provides an explanation for each command:

- `UltraMappedInterval(a, b, k)` creates a mapped interval $[a, b]$ with a Gauss–Jacobi grid determined by the ultraspherical parameter k.
- `Ultraspherical(`λ`, UltraMappedInterval(a, b, k))` constructs a ultraspherical polynomial basis with parameter λ on the mapped interval $[a, b]$, using a Gauss–Jacobi grid determined by k.
- `PeriodicMappedInterval(a, b)` defines a periodic grid mapping over the interval $[a, b]$.
- `Fourier(PeriodicMappedInterval(a, b))` defines a Fourier basis over the interval $[a, b]$ with periodic boundary conditions.
- `RationalRealAxis()` defines a mapped real line domain using an inverse Möbius transform that maps the unit circle to the real axis
- `OscRational(RationalRealAxis(), a)` creates a rational basis on the real line (via RationalRealAxis) with oscillatory behavior controlled by phase shift parameter `a`.
- `Derivative(k)` defines an abstract kth-order differentiation operator.

- `WeightedDerivative(OscRational(RationalRealAxis(), a))` builds an upper-diagonal operator that maps each mode R_j to jR_{j-1}, used to represent $xf'(x)$ in the rational basis (only valid for `a`$= 0$).

- `Multiplication(x -> f(x))` defines an abstract operator representing the multiplication by some function $f(x)$.

- `Conversion(◊)` defines an abstract conversion operator that maps the input function space to the target space ◊ upon which it acts.

- `GridValues(UltraMappedInterval(a, b, k))` defines a nonuniform evaluation basis over $[a, b]$ using Gauss–Jacobi points controlled by parameter k, without fixing the number of points.

- `FixedGridValues([m], UltraMappedInterval(a, b, k))` creates a discrete basis that evaluates functions at $m \in [a, b]$, using the coordinate transformation defined by `UltraMappedInterval(a, b, k)`.

- `BasisExpansion(x -> f(x), ◊, n)` returns the basis expansion of the function $f(x)$ using the basis ◊ with n coefficients. Moreover, `BasisExpansion(x -> f(x), ◊, n).c` returns the coefficient vector.

- `BasisExpansion(◊, coeff)` returns a function using the given coefficient vector `coeff` and the basis ◊

The package `ApproxFun.jl` is similar to `OperatorApproximation.jl` except that the coefficients are not normalized and the matrices can be obtained by directly truncating the defined operators. The following list provides an explanation for each command:

- `x = Fun()` defines the identity function $x \to x$ on $[-1, 1]$ in the Chebyshev basis.

- `Chebyshev()` Constructs the default Chebyshev polynomial basis on the interval $[-1, 1]$.

- `Fun(x -> f(x), ◊, n)` Approximates the function $f(x)$ using n basis functions from the space ◊ (e.g., Chebyshev(), Fourier(), etc.), producing a spectral representation of f. `Fun(x -> f(x), ◊, n).coefficients` returns the coefficient vector.

- `Derivative()` creates the abstract 1st-order differentiation operator.

- `Derivative():Chebyshev()` constructs the concrete operator representing the 1st-order differentiation operator in the Chebyshev basis.
- `(Derivative():Chebyshev())`^{k} constructs the concrete operator representing the kth-order differentiation operator in the Chebyshev basis.
- `Multiplication(f(x), ◊)` constructs a linear operator representing the multiplication by a function $f(x)$ in the basis ◊.
- `Conversion(◊, □)` defines a conversion operator from the basis ◊ to the basis □.

Bibliography

[Arn51] W Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics **9** (1951), 17–29.

[Bas83] F Bashforth, *An attempt to test the theories of capillary action by comparing the theoretical and measured forms of drops of fluid with an explanation of the method of integration employed in constucting the tables which give the theoretical forms of such drops*, Cambridge University Press (1883), 1819–1892.

[Blo11] A Bloemendal, *Finite Rank Perturbations of Random Matrices and Their Continuum Limits*, Ph.D. thesis, University of Toronto, 2011.

[Bor09] F Bornemann, *On the Numerical Evaluation of Distributions in Random Matrix Theory: A Review*, 4 2009.

[Boy01] J.P. Boyd, *Chebyshev and Fourier Spectral Methods*, second ed., Dover Books on Mathematics, Dover Publications, Mineola, NY, 2001.

[BRF83] R.E. Bank, D.J. Rose, and W Fichtner, *Numerical methods for semiconductor device simulation*, IEEE Transactions on Electron Devices **30** (1983), 1031–1041.

[BS08] S.C. Brenner and L.R. Scott, *The Mathematical Theory of Finite Element Methods*, vol. 15, Springer New York, 2008.

[BT97] G Baszenski and M Tasche, *Fast polynomial multiplication and convolutions related to the discrete cosine transform*, Linear Algebra and its Applications **252** (1997), 1–25.

[BV13] A Bloemendal and B Virág, *Limits of spiked random matrices I*, Probability Theory and Related Fields **156** (2013), 795–825.

[CG03] S Chandrasekaran and M Gu, *Fast and Stable Algorithms for Banded Plus Semiseparable Systems of Linear Equations*, SIAM Journal on Matrix Analysis and Applications **25** (2003), 373–384.

[CH52] C.F. Curtiss and J.O. Hirschfelder, *Integration of Stiff Equations*, Proceedings of the National Academy of Sciences **38** (1952), 235–243.

[Dah56] G Dahlquist, *Convergence and stability in the numerical integration of ordinary differential equations*, Mathematica Scandinavica **4** (1956), 33–53.

[Dah63] ———, *A special stability problem for linear multistep methods*, BIT **3** (1963), 27–43.

[DE02] I Dumitriu and A Edelman, *Matrix models for beta ensembles*, Journal of Mathematical Physics **43** (2002), 5830–5847.

[DH16] T.A. Driscoll and N Hale, *Rectangular spectral collocation*, IMA Journal of Numerical Analysis **36** (2016), 108–132.

[ES07] A Edelman and B Sutton, *From Random Matrices to Stochastic Operators*, Journal of Statistical Physics **127** (2007), 1121–1165.

[GIKM16] T Grava, A Its, A Kapaev, and F Mezzadri, *On the Tracy-Widom β Distribution for β=6*, Symmetry, Integrability and Geometry: Methods and Applications (2016), 26 pages.

[Giv58] W Givens, *Computation of Plain Unitary Rotations Transforming a General Matrix to Triangular Form*, Journal of the Society for Industrial and Applied Mathematics **6** (1958), 26–50.

[GL13] G Golub and C Van Loan, *Matrix Computations*, 4 ed., Johns Hopkins University Press, Philadelphia, PA, 2013.

[Gre97] A Greenbaum, *Iterative Methods for Solving Linear Systems*, Society for Industrial and Applied Mathematics, 1997.

[HW96] E Hairer and G Wanner, *Solving Ordinary Differential Equations ii*, vol. 14, Springer Berlin Heidelberg, 1996.

[Jac11] D Jackson, *Über die Genauigkeit der annäherung stetiger Funktionen durch ganze rationale Funktionen gegebenen Grades und trigonometrische Summen gegebener Ordnung*, Dieterich, 1911.

[Jac13] ———, *On the Accuracy of Trigonometric Interpolation*, Transactions of the American Mathematical Society **14** (1913), no. 4, 453–461.

[Kur78] Y Kuramoto, *Diffusion-induced chaos in reaction systems*, Progress of Theoretical Physics Supplement **64** (1978), 346–367.

[Kut01] W. Kutta, *Beitrag zur näherungsweisen Integration totaler Differentialgleichungen*, Zeit. Math. Phys. **46** (1901), 435–453.

[Lan38] C Lanczos, *Trigonometric Interpolation of Empirical and Analytical Functions*, Journal of Mathematics and Physics **17** (1938), 123–199.

[LeV07] R LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*, Society for Industrial and Applied Mathematics, 2007.

[Li18] Y Li, *On the Open Question of The Tracy-Widom Distribution of β-Ensemble With $\beta = 6$*, 2018.

[Meh04] M Mehta, *Random Matrices*, 3rd ed., Academic Press, New York, 2004.

[Mou26] F.R. Moulton, *New methods in exterior ballistics*, University of Chicago Press, Chicago, IL, 1926.

[MS00] H Ma and W Sun, *A legendre–petrov–galerkin and chebyshev collocation method for third-order differential equations*, SIAM Journal on Numerical Analysis **38** (2000), 1425–1438.

[OGS+15] S Olver, G Goretkin, R Slevinsky, A Townsend, et al., https://github.com/JuliaApproximation/ApproxFun.jl, 2015.

[Ors72] S.A. Orszag, *Comparison of Pseudospectral and Spectral Approximation*, Studies in Applied Mathematics **51** (1972), 253–259.

[Ose11] I.V. Oseledets, *Tensor-Train Decomposition*, SIAM Journal on Scientific Computing **33** (2011), 2295–2317.

[OT13] S Olver and A Townsend, *A fast and well-conditioned spectral method*, SIAM Review **55** (2013), 462–489.

[PK90] T.N. Phillips and A Karageorghis, *On the Coefficients of Integrated Expansions of Ultraspherical Polynomials*, SIAM Journal on Numerical Analysis **27** (1990), 823–830.

[Rev15] J Revels, https://github.com/JuliaCI/BenchmarkTools.jl, 2015.

[RRV11] J Ramírez, B Rider, and B Virág, *Beta ensembles, stochastic Airy spectrum, and a diffusion*, Journal of the American Mathematical Society **24** (2011), 919–944.

[Rum16] I Rumanov, *Painlevé Representation of Tracy–Widom$_\beta$ Distribution for $\beta = 6$*, Communications in Mathematical Physics **342** (2016), no. 3, 843–868.

[Run95] C Runge, *Ueber die numerische Auflösung von Differentialgleichungen*, Mathematische Annalen **46** (1895), 167–178.

[Saa03] Y Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, 2003.

[She95] J Shen, *Efficient spectral-galerkin method ii. direct solvers of second- and fourth-order equations using chebyshev polynomials*, SIAM Journal on Scientific Computing **16** (1995), 74–87.

[Siv77] G.I. Sivashinsky, *Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations*, Acta Astronautica **4** (1977), 1177–1206.

[Siv80] ———, *On flame propagation under conditions of stoichiometry*, SIAM Journal on Applied Mathematics **39** (1980), 67–82.

[SS86] Y Saad and M Schultz, *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*, SIAM Journal on Scientific and Statistical Computing **7** (1986), 856–869.

[Sut06] B Sutton, *The Stochastic Operator Approach to Random Matrix Theory*, Ph.D. thesis, Massachusetts Institute of Technology, 2006.

[TL24] T Trogdon and K Lilly, https://github.com/tomtrogdon/OperatorApproximation.jl, 2024.

[TO15] A Townsend and S Olver, *The automatic solution of partial differential equations using a global spectral method*, Journal of Computational Physics **299** (2015), 106–123.

[Tow14] A Townsend, *Computing with functions in two dimensions*, Ph.D. thesis, University of Oxford, 2014.

[Tre00] L.N. Trefethen, *Spectral methods in matlab*, Society for Industrial and Applied Mathematics, 2000.

[Tro24] T Trogdon, *The ultraspherical rectangular collocation method and its convergence*, 2024.

[TW93] C Tracy and H Widom, *Level-spacing distributions and the Airy kernel*, Physics Letters B **305** (1993), no. 1, 115–118.

[TW94] ———, *Level-spacing distributions and the Airy kernel*, Communications in Mathematical Physics **159** (1994), 151–174.

[TW96] ———, *On orthogonal and symplectic matrix ensembles*, Communications in Mathematical Physics **177** (1996), no. 3, 727–754.

[TZ24] T Trogdon and Y Zhang, *Computing the Tracy-Widom Distribution for Arbitrary* $\beta > 0$, Symmetry, Integrability and Geometry: Methods and Applications (2024).

[VV09] B Valkó and B Virág, *Continuum limits of random matrices and the brownian carousel*, Inventiones mathematicae **177** (2009), 463–508.

[Wer80] A Werschulz, *Computational complexity of one-step methods for systems of differential equations*, Mathematics of Computation **34** (1980), 155–174.

[Wid67] O.B. Widlund, *A note on unconditionally stable linear multistep methods*, BIT **7** (1967), 65–70.

[Woo50] M Woodbury, *Inverting modified matrices*, Memorandum Rept. 42, Statistical Research Group, Princeton Univ., 1950.