ADAPTIVE PARTITION OF UNITY METHODS FOR CHEBYSHEV POLYNOMIAL APPROXIMATION AND NONLINEAR ADDITIVE SCHWARZ PRECONDITIONING

by

Kevin Aiton

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Applied Mathematics

Summer 2019

ADAPTIVE PARTITION OF UNITY METHODS FOR CHEBYSHEV POLYNOMIAL APPROXIMATION AND NONLINEAR ADDITIVE SCHWARZ PRECONDITIONING

by

Kevin Aiton

Approved: _____

Louis Rossi, Ph.D.
Chair of the Department of Mathematical Sciences

Approved: _____

John Pelesko, Ph.D.
Dean of the College of Arts and Sciences

Approved: _____

Douglas J. Doren, Ph.D.
Interim Vice Provost for Graduate and Professional Education

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Tobin Driscoll, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Richard Braun, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Louis Rossi, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Rodrigo Platte, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

I would like to start by expressing my gratitude to my advisor Professor Tobin Driscoll. I thank you for your constant support and encouragement throughout the years. I've found your pragmatic approach to research refreshing. I especially appreciate the autonomy you've given me in my research. It has allowed me to grow into a researcher in my own right.

Thank you Professor Braun for supporting me within the tear film modeling group, in particular agreeing to fund me as a research assistant. I appreciate the encouragement you've given me to present and share my research. It has been a pleasure collaborating with you. I'd also like to thank Professor Rossi and Platte for being on my committee, and helping me revise this dissertation.

The friends I've made while finishing my dissertation have significantly enriched my life. I'd like to thank my academic sisters Amy, Lan and Rayanne. I'd especially like to thank Matt, Zach and Lan for the fun times we've shared in the math department.

I'd like to thank my family for supporting me while finishing my dissertation. My parents have helped me anytime that I have asked. I would have not pursued a PhD without their strong encouragement. It was my father that pushed me to move forward with my academic career. My siblings Emily, Scott and Mitch have been very supportive.

I especially thank Moe for the love and experiences we've shared together. I feel fortunate to have you in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

If a function is analytic on and around an interval, then Chebyshev polynomial interpolation provides spectral convergence. However, if the function has a singularity close to the interval, the rate of convergence is near unity. In these cases splitting the interval and using piecewise interpolation can accelerate convergence. Chebfun includes a splitting mode that finds an optimal splitting through recursive bisection, but the result has no global smoothness unless conditions are imposed explicitly at the breakpoints. We developed a new technique where we split the domain into overlapping intervals and use an infinitely smooth partition of unity to blend the local Chebyshev interpolants.

We construct the partition of unity approximations with a simple divide-and-conquer algorithm similar to Chebfun's splitting mode can be used to find an overlapping splitting adapted to features of the function. Our algorithm implicitly constructs the partition of unity over the subdomains, without the need of explicitly keeping track of neighboring subdomains. This allows us to use a partition of unity method within an adaptive frame work. We applied this technique explicitly on given functions as well as to the solutions of singularly perturbed boundary value problems.

The extension of the Chebfun technique to two-dimensional and three-dimensional functions on hyperrectangles has mainly focused on low-rank approximation. While this method is very effective for some functions, it is highly anisotropic and unacceptably slow for many functions of potential interest. We developed a method based on automatic recursive domain splitting, with a partition of unity to define the global approximation that is easy to construct and manipulate. Our experiments show it to be as fast as existing software for many low-rank functions, and much faster on other examples, even in serial computation. It is also much less sensitive to alignment with

coordinate axes. We utilized the tree structure to develop fast algorithms for interpolation, integration and differentiation. In particular, we developed a fast and efficient scheme for arithmetically combining partition of unity approximations using the tree representations that would have not been possible with approximations represented with graphs. We took steps to develop approximations of functions on nonrectangular domains, by using least-squares polynomial approximations in a manner similar to Fourier extension methods, with promising results.

The additive Schwarz method is usually presented as a preconditioner for a PDE linearization based on overlapping subsets of nodes from a global discretization. It has previously been shown how to apply Schwarz preconditioning to a nonlinear problem using subdomains with a shared global grid. We expand on these ideas by first replacing the original global PDE with the Schwarz overlapping problem, where each subdomain has an independent grid, ~~where~~ unknowns do not need to be shared. This allows us to avoid restrictive-type updates since subdomains need to communicate only via interface interpolations. Our new preconditioner can be applied linearly or nonlinearly. In the latter case we solve nonlinear subdomain problems independently in parallel. With our new nonlinear preconditioned method, the frequency and amount of interprocess communication is greatly reduced compared to linearized preconditioning. Our method allows us to adapt to the features of a PDE solution, ~~which is necessary for~~ the numerical solutions of fourth order tear film models, which are both stiff and highly nonlinear.

Chapter 1

INTRODUCTION

A distinctive and powerful mode of scientific computation has emerged recently in which mathematical functions are represented by high-accuracy numerical analogs, which are then manipulated or analyzed numerically using a high-level toolset [69]. The most prominent example of this style of computing is the open-source Chebfun project [8, 23]. Chebfun, which is written in MATLAB, samples a given piecewise-smooth univariate function at scaled Chebyshev nodes and automatically determines a Chebyshev polynomial interpolant for the data, resulting in an approximation that is typically within a small multiple of double precision of the original function. This approximation can then be operated on and analyzed with algorithms that are fast in both the asymptotic and real-time senses. Notable operations include rootfinding, integration, optimization, solution of initial- and boundary-value problems, eigenvalues of differential and integral operators, and solution of time-dependent PDEs [36, 58, 7].

The main aim of Chebfun is to allow for computation that feels symbolic, but is performed numerically in a way that is both fast and accurate, similar to what floating-point arithmetic achieves for numbers [70]. Chebfun used Chebyshev polynomial interpolants because they are highly accurate and stable even for large amounts of points [51, 72]. In particular if we have a Chebyshev interpolant $p_n(x)$ of a analytic function $f(x)$ then the Chebyshev interpolants are spectrally accurate, as stated in Theorem 6 of [68]:

Theorem 1.0.1. *Suppose $f(z)$ is analytic on and inside the Bernstein ellipse E_ρ. Let p_n be the polynomial that interpolates $f(z)$ at $n+1$ Chebyshev points of the second*

kind. Then there exists a constant $C > 0$ such that for all $n > 0$,

$$\|f(x) - p_n(x)\|_\infty \leq C\rho^{-n}.$$

If $f(x)$ is Lipschitz continuous on $[-1, 1]$ then

$$f(x) = \sum_{k=0}^{\infty} a_k T_k(x), \quad a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx, \tag{1.1}$$

where T_k denotes the degree k Chebyshev polynomial (and for a_0, we multiply by $\frac{1}{\pi}$ instead of $\frac{2}{\pi}$). Furthermore if $p_n(x)$ is the nth degree Chebyshev interpolant then

$$f(x) - p_n(x) = \sum_{k=n+1}^{\infty} a_k \left(T_k(x) - T_m(x)\right), \tag{1.2}$$

where

$$m = \left[(k + n - 1)(\text{mod } 2n) - (n - 1)\right], \tag{1.3}$$

implying we can determine the accuracy of the interpolant $p_n(x)$ by inspecting the Chebyshev coefficients [72]. Chebfun's `standardChop` method determines the minimum required degree by searching for a plateau of low magnitude coefficients [5]. For example, Figure 1.1 shows the first 128 coefficients of $f(x) = \exp(\sin(\pi x))$. We see that all coefficients after the first 46 have magnitude less than 10^{-15}. In this case, Chebfun determines the ideal degree to be 50.

Townsend and Trefethen extended the 1D Chebfun algorithms to 2D functions over rectangles in Chebfun2 [67, 66], which uses low-rank approximations in an adaptive cross approximation. The construction and manipulation of 2D approximations is suitably fast for a wide range of smooth examples. Most recently, Hashemi and Trefethen created an extension of Chebfun called Chebfun3 for 3D approximations on hyperrectangles using low-rank "slice–Tucker" decompositions [37]. The range of functions that Chebfun3 can cope with in a reasonable interactive computing time is somewhat narrower than for Chebfun2, as one would expect.

An alternative to Chebfun and related projects ported to other languages is sparse grid interpolation. Here one uses linear or polynomial interpolants on hierarchical Smolyak grids. Notable examples of software based on this technique are the Sparse

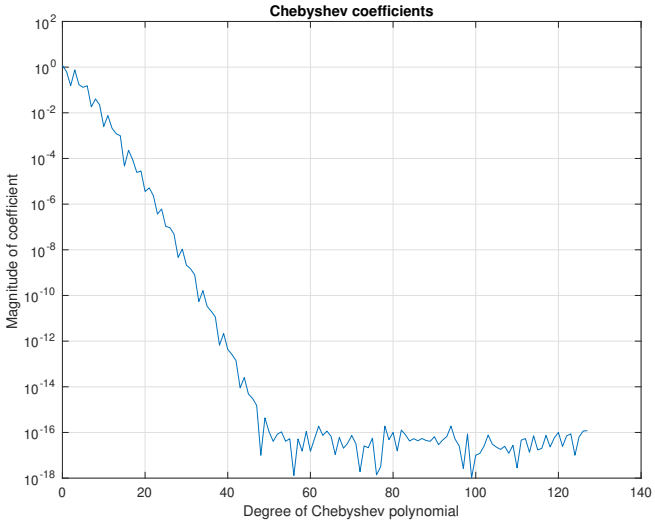


Figure 1.1: Chebyshev coefficients for $f(x) = \exp(\sin(\pi x))$.

Grid Interpolation Toolbox [44] and the Sparse Grids Matlab Kit [6]. An advantage of these packages is that they are capable of at least medium-dimensional representations on hyperrectangles. However, they seem to be less focused on high-accuracy approximation for a wide range of functions, and they are less fully featured than the Chebfun family. These methods are also highly nonisotropic.

The more general problem of approximation of a function with high pointwise accuracy over a nonrectangular domain $\Omega \subset \mathbb{R}^d$ allows more limited global options than in the hyperrectangular case. Neither low-rank nor sparse grid approximations have any clear global generalizations to this case. Two techniques that can achieve spectral convergence for at least some such domains are radial basis functions [29] and Fourier extension or continuation [1], but neither has been conclusively demonstrated to operate with high speed and reliability over a large collection of domains and functions.

In this dissertation, we expand on the ideas of Chebfun to create approximations that are both accurate and infinitely smooth via partition of unity, a set of functions whose sum must add to one. Partition of unity schemes have been widely used for interpolation [31, 53, 63] and solving PDEs [35, 60]. Suppose we have an overlapping covering $\{\Omega_k\}_{k=1}^N$ on a bounded region Ω. A partition of unity is a collection of real valued functions $\{w_k(\mathbf{x})\}_{k=1}^N$ such that:

- $w_k(\mathbf{x})$ has support within Ω_k,
- each $w_k(\mathbf{x})$ is nonnegative,
- $\forall x \in \Omega, \quad \sum_{k=1}^{N} w_k(\mathbf{x}) = 1.$

The functions $\{w_k(\mathbf{x})\}_{k=1}^{N}$ are called the *weights* of the partition. We can use the partition of unity $\{w_k(\mathbf{x})\}_{k=1}^{N}$ to construct an approximating function. Suppose that for $m \geq 0$ we have a function $f \in C^m(\Omega)$, each weight $w_k(\mathbf{x}) \in C^m(\Omega)$ and for each patch Ω_k we have an approximation $s_k(\mathbf{x})$ of $f(\mathbf{x})$. Then the function

$$s(\mathbf{x}) = \sum_{k=1}^{N} w_k(\mathbf{x}) s_k(\mathbf{x}) \tag{1.4}$$

can be used to approximate $f(\mathbf{x})$ and its derivatives [73].

Theorem 1.0.2. *Suppose $f \in C^m(\Omega)$ and for each patch Ω_k we have a function $s_k(\mathbf{x})$ such that*

$$\|D^\alpha f(\mathbf{x}) - D^\alpha s_k(\mathbf{x})\|_{L_\infty(\Omega_k)} \leq \varepsilon_k(\alpha), \quad \mathbf{x} \in \Omega$$

for all $|\alpha| \leq k$. Thus for $j \leq m$, if $s(x)$ is the approximation (1.4) then

$$\|D^\alpha f(\mathbf{x}) - D^\alpha s_k(\mathbf{x})\|_{L_\infty(\Omega_k)} \leq \sum_{k=1}^{N} \sum_{\beta \leq \alpha} \binom{\alpha}{\beta} \left\|D^{\alpha-\beta} w_k(\mathbf{x})\right\|_{L_\infty(\Omega_k)} \epsilon_k(\alpha). \tag{1.5}$$

Proof. Since $\sum_{k=1} w_k(\mathbf{x}) = 1$, $\sum_{k=1} w_k(\mathbf{x}) f(\mathbf{x}) = f(\mathbf{x})$. Thus

$$\begin{aligned} D^\alpha f(\mathbf{x}) - D^\alpha s_k(\mathbf{x}) &= D^\alpha \sum_{k=1}^{N} w_k(\mathbf{x})(f(\mathbf{x}) - s_k(\mathbf{x}x)) \\ &= \sum_{k=1}^{N} \sum_{\beta \leq \alpha} \binom{\alpha}{\beta} D^{\alpha-\beta} w_k(\mathbf{x}) \left(D^\beta f(\mathbf{x}) - s_k^{(i)}(\mathbf{x})\right). \end{aligned} \tag{1.6}$$

The result follows from here by the triangle inequality. □

A major goal of Chebfun has been to develop methods to automatically find solutions to ODEs using Chebyshev approximations [24, 10, 9]. We build on our overlapping domain approximations to solve boundary-value and evolutionary PDEs efficiently via domain decomposition methods. Overlapping domain decomposition has

been recognized as a valuable aid in solving partial differential equations since Schwarz first described his alternating method in 1870. (For straightforward introductions to the topic, see [22, 64]; for a more historical perspective, see [32].) Overlapping decomposition provides a way to solve a problem on a global domain by exploiting its reduction to smaller subdomains. This creates geometric flexibility and allows special effort to be focused on small parts of the domain when appropriate. Domain decomposition also has a natural parallelism that is particularly attractive in the increasingly multicore context of scientific computing.

For a linear PDE, one typically seeks to apply a preconditioner for a Krylov iteration such as GMRES, in the form of solving problems on overlapping subdomains whose boundary data is in part determined by values of the solution in other subdomains. In the parallel context this is achieved by an additive Schwarz (AS) scheme. When one partitions the unknowns of a global discretization into overlapping subsets, the best form of AS are restricted AS (RAS) methods [16], which do not allow multiple domains to update shared unknowns independently and thus over-correct. Typically, then, the subdomain problems are solved on overlapping sets, but the results are distributed in a nonoverlapping fashion.

Cai and Keyes [15] proposed instead modifying the *nonlinear* problem using the Schwarz ansatz. In addition to yielding a preconditioned linearization for the Krylov solver, the preconditioned nonlinear problem exhibited more robust convergence for the Newton iteration than did the original nonlinear problem. They called their method ASPIN, short for *additive Schwarz preconditioned inexact Newton*. Subsequently, Dolean et al. [21] pointed out that Cai and Keyes did not use the RAS form of AS preconditioning, and they proposed an improved variant called RASPEN that does.

A major motivation for our focus on domain decomposition methods is for the numerical solutions of a fourth order blinking eye model. In the past, on overset grid method has been used to solve models on a realistic eye shaped domain [17, 38, 46, 45]. The overset grid method discritizes the PDE by covering the domain with overlapping

finite difference grids coupled together through interpolation. This method, while highly effective, is limited in accuracy. Driscoll and Braun developed a method to simulate parabolic flow on a blinking eye shaped domain, which was discretized using a tensor product Chebyshev polynomial approximation [25]. While the use of a spectral approximation led to higher accuracy, tensor product polynomials are expensive to use computationally since the matrices representing the PDE discritizations are dense.

1.1 Contributions of this dissertation

In Chapter 2, we develop approximations that are adapted to the features of the function, all while being infinitely continuous. With the use of a partition of unity, we can blend local approximations together without matching. We show how in one dimension the partition of unity weights can be defined recursively. We finish by showing how these approximations can be used to solve BVPs.

We next extend this approach in Chapter 3 to construct adaptive approximations on hyper-rectangles in any dimension. We developed a multitude of algorithms to interact with them. Our new method exhibits far less anisotropy than the approximations of Chebfun2 and Chebfun3. We extend our technique to non-rectangular domains via a least squares methods, and show it is competitive with existing Fourier extension methods.

For Chapter 4 we expand on the ideas of Cai, Keyes and Dolean to develop a new nonlinear preconditioned method based on the Alternating Schwarz technique. We alternatively use separate grids for the overlapping domains (as apposed to a globally shared one). This allows us to more easily adapt the subdomains to the features of the underlying BVP solution. Nonlinear preconditioning produces a method that is robust in the face of increasing non-linearity.

We lastly in Chapter 5 apply our new preconditioned techniques on a fourth order model that represents the tear film fluid within a moving eye-shaped domain. Our new method allows us to achieve the resolution needed to preserve the volume

over a blink cycle. We preform numerical experiments on the current model, and give guidance for how it can be further refined.

Chapter 2

AN ADAPTIVE PARTITION OF UNITY METHOD FOR ONE DIMENSIONAL CHEBYSHEV POLYNOMIAL INTERPOLATION

Chebfun includes a *splitting* algorithm that creates piecewise polynomial approximations [57]. When splitting is enabled, if a Chebyshev interpolant is unable to represent the function accurately at a specified maximum degree on an interval, the interval is bisected; this process is recursively repeated on the subintervals. Afterwards adjacent subintervals are merged if the new interval allows for a Chebyshev approximation with lower degree. In effect, the method does a binary search for a good splitting locations. In [27] it was shown that the splitting locations are roughly optimal based on the singularity structure of the function in the complex plane.

A drawback of Chebfun's splitting approach is that the resulting representation does not ensure anything more than C^0 continuity. Differentiation of the Chebyshev interpolating polynomial of degree n has norm $O(n^2)$, so a jump in the derivative develops across a splitting point and becomes more pronounced for higher derivatives and larger n. In order to solve a boundary-value problem, Chebfun imposes explicit continuity conditions on the solution to augment the discrete problem. This solution works well in 1D but becomes cumbersome in higher dimensions, particularly if refinements are made nonconformingly.

In this chapter we explore the use of Chebyshev interpolants on overlapping domains combined using a *partition of unity*. The resulting approximation has the same accuracy as the individual piecewise interpolants. We use compactly supported weight functions that are infinitely differentiable, so the resulting combined interpolant is also infinitely smooth (though not analytic). We also show that the accuracy of the derivative can be bounded by $\Theta(\delta^{-2})$ for an overlap amount δ, revealing an explicit

tradeoff between efficiency (smaller overlap and more like Chebfun splitting) and global accuracy of the derivative. Because the global approximation is smooth, there are no matching conditions needed to solve a BVP, and there are standard preconditioners available that aid with iterative methods for large discretizations. For example, since we split the interval into overlapping domains we could use the restricted additive Schwarz preconditioner as we do in Chapter 4.

We describe a recursive, adaptive algorithm for creating and applying a partition of unity, modeled on the recursive splitting in Chebfun but merging adjacent subdomains aggressively in order to keep the total node count low. Even though each node of the recursion only combines two adjacent subdomains, we show that the global approximant is also a partition of unity. We demonstrate that the adaptive refinement is able to resolve highly localized features of an explicitly given function and of a solution to a singularly perturbed BVP.

In section 2.1 we discuss the convergence of the method for a simple split of the interval $[-1, 1]$. We describe our adaptive algorithm in section 2.2. In section 2.3 we explain how to apply our method to solve boundary value problems on an interval and perform some experiments with singularly perturbed problems.

2.1 Convergence analysis

In this section we consider a single interval partitioned into two overlapping parts, i.e. $[-1, t]$,$[-t, 1]$, where t is the overlap parameter such that $0 < t < 1$. For the weights, we use Shepard's method [63] based on the compactly supported, infinitely differentiable shape function

$$\psi(x) = \begin{cases} \exp\left(1 - \frac{1}{1-x^2}\right) & |x| < 1, \\ 0 & |x| \geq 1. \end{cases} \tag{2.1}$$

We define support functions

$$\psi_\ell(x) = \psi\left(\frac{x+1}{1+t}\right) \quad \text{and} \quad \psi_r(x) = \psi\left(\frac{x-1}{1+t}\right), \tag{2.2}$$

to construct the PU weight functions

$$w_\ell(x) = \frac{\psi_\ell(x)}{\psi_\ell(x) + \psi_r(x)} \quad \text{and} \quad w_r(x) = \frac{\psi_r(x)}{\psi_\ell(x) + \psi_r(x)}, \tag{2.3}$$

where $w_\ell(x), w_r(x)$ have support on the left and right intervals, respectively.

Suppose that $s_\ell(x), s_r(x)$ approximate $f(x)$ on $[-1, t]$, $[-t, 1]$ respectively and are both infinitely smooth. Let

$$s(x) = w_\ell(x)s_\ell(x) + w_r(x)s_r(x), \tag{2.4}$$

where $s(x)$ is the partition of unity approximation. Following Theorem 1.0.2 we have for $x \in [-1, 1]$ that

$$\begin{aligned} |f(x) - s(x)| &= |w_\ell(x)\,(f(x) - s_\ell(x)) + w_r(x)\,(f(x) - s_r(x))| \\ &\leq w_\ell(x)\,|f(x) - s_\ell(x)| + w_r(x)\,|f(x) - s_r(x)|\,. \end{aligned} \tag{2.5}$$

We conclude that

$$\|f(x) - s(x)\|_{L_\infty[-1,1]} \leq \max\left(\|f(x) - s_\ell(x)\|_{L_\infty[-1,t]}\,, \|f(x) - s_r(x)\|_{L_\infty[-t,1]}\right). \tag{2.6}$$

This implies that the partition of unity method (PUM) preserves the accuracy of its local approximants. We also have that $s(x)$ is infinitely smooth. For the first derivative we have

$$\begin{aligned} |f'(x) - s'(x)| \leq\ & |w_\ell(x)\,(f'(x) - s'_\ell(x))| + |w_r(x)\,(f'(x) - s'_r(x))| \\ & + |w'_\ell(x)\,(f(x) - s_\ell(x))| + |w'_r(x)\,(f(x) - s_r(x))|\,, \end{aligned} \tag{2.7}$$

giving us

$$\begin{aligned} \|f'(x) - s'(x)\|_{L_\infty[-1,1]} \leq\ & \max\left(\|f'(x) - s'_\ell(x)\|_{L_\infty[-1,t]}\,, \|f'(x) - s'_r(x)\|_{L_\infty[-t,1]}\right) \\ & + \|w'_\ell(x)\|_{L_\infty[-t,t]}\,\|f(x) - s_\ell(x)\|_{L_\infty[-t,t]} \\ & + \|w'_r(x)\|_{L_\infty[-t,t]}\,\|f(x) - s_r(x)\|_{L_\infty[-t,t]}\,, \end{aligned} \tag{2.8}$$

since the derivatives of the weights have support only on the overlap. For $t \ll 1$, the weights steepen to become nearly step functions. This causes the derivatives of the weights to be large in magnitude, resulting in an increase in the error for the derivative.

Since $w'_\ell(x) = -w'_r(x)$, from (2.8) we can infer

$$\begin{aligned}\|f'(x) - s'(x)\|_{L_\infty[-1,1]} \leq \max\left(\|f'(x) - s'_\ell(x)\|_{L_\infty[-1,t]}, \|f'(x) - s'_r(x)\|_{L_\infty[-t,1]}\right) \\ + \|w'_\ell(x)\|_{L_\infty[-t,t]} \max\left(\|f(x) - s_\ell(x)\|_{L_\infty[-t,t]}, \|f(x) - s_r(x)\|_{L_\infty[-t,t]}\right).\end{aligned} \tag{2.9}$$

We have that $x = 0$ is a critical point of $w'_\ell(x)$ and for $t < 0.4$ it can be shown that the maximum of $|w'_\ell(x)|$ occurs at $x = 0$. Since

$$w'_\ell(0) = -\frac{(1+t)^2}{t^2(2+t)^2}, \tag{2.10}$$

we can infer that $\|w'_\ell(x)\|_{L_\infty[-t,t]} = \Theta(t^{-2})$ as $t \to 0$. The norm of the Chebyshev differentiation operator is $\Theta(n^2)$ (for n nodes), implying that the two terms on the right-hand side of (2.9) are balanced if $t^{-2} = \Theta(n^2)$, or equivalently $t = \Theta\left(\frac{1}{n}\right)$. A simple example of a split can be seen in Figure 2.1.

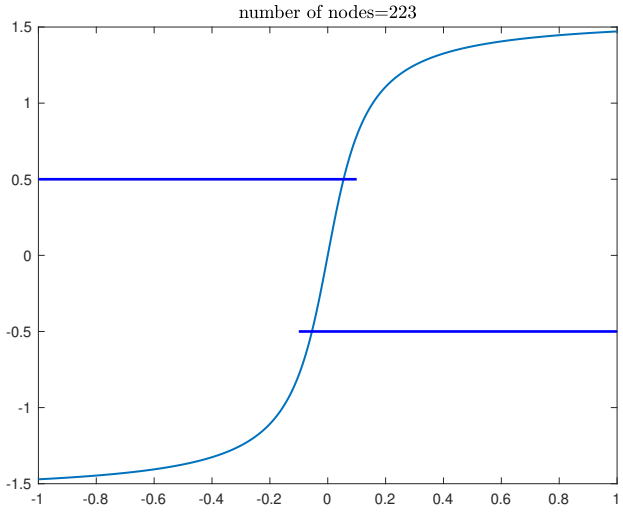


Figure 2.1: Plot of the PU approximation with overlap parameter $t = 0.1$ for $f(x) = \arctan(x/0.1)$, where the thick lines represent the domains of the left and right approximation. Here $\|f(x) - s(x)\|_{L_\infty[-1,1]} = 2.4\text{e}{-15}$ and $\|f'(x) - s'(x)\|_{L_\infty[-1,1]} = 1.7\text{e}{-13}$.

2.2 Recursive algorithm

In order to allow for adaptation to specific features of $f(x)$, we next describe a recursive bisection algorithm that works similarly to Chebfun's splitting algorithm [27] and to that of [65]. Suppose we want to construct a PU approximation $s_{[a,b]}(x)$

on the interval $[a, b]$ using Chebyshev interpolants on the patches. Our strategy is to cover $[a, b]$ with overlapping subdomains, on each of which f is well-approximated by a Chebyshev polynomial, and using the binary tree recursively define a partition of unity approximation.

In this section we describe an adaptive procedure for obtaining the overlapping domains and individual approximations over them. The domains are constructed from recursive bisections of $[a, b]$ into nonoverlapping interverals, referred to as *zones*. Given a zone $[\alpha, \beta]$, we extend it to a larger domain $[\bar{\alpha}, \bar{\beta}]$ by fixing a parameter $t > 0$, defining

$$\delta_j = \frac{\beta - \alpha}{2}(1 + t), \tag{2.11}$$

and then setting

$$\bar{\alpha} = \max\{a, \beta - \delta\}, \quad \bar{\beta} = \min\{\alpha + \delta, b\}. \tag{2.12}$$

In words, the zone is extended on both sides by an amount proportional, up to the boundary of the global domain $[a, b]$.

We define a binary tree T with each node ν having the following properties:

- $\mathsf{domain}(\nu)$:=the domain of the patch
- $\mathsf{zone}(\nu)$:=the zone of the patch
- $\mathsf{child}_0(\nu)$,$\mathsf{child}_1(\nu)$:=respective left and right subtrees of ν (if split)
- $\mathrm{w}_0(\nu)$,$\mathrm{w}_1(\nu)$:=respective left and right weights of ν (if split)
- $\mathsf{interpolant}(\nu)$:=Chebyshev interpolant on $\mathsf{domain}(\nu)$ if ν is a leaf
- $\mathsf{values}(\nu)$:=values of the function we are approximating at the Chebyshev points of ν.

For a node ν we recursively define the approximation on $\mathsf{domain}(\nu)$ as

$$s_\nu(x) = w_0(\nu) s_{\mathsf{child}_0(\nu)}(x) + w_1(\nu) s_{\mathsf{child}_1(\nu)}(x) \tag{2.13}$$

where approximations on leaves are defined to be Chebyshev polynomials. We prove in Theorem 2.2.1 that this results in a partition of unity approximation. In Algorithm 1

Algorithm 1 refine(ν,f,N,t)

if ν is a leaf and f cannot be resolved by $\mathsf{interpolant}(\nu)$ **then**
  Define new nodes ν_0, ν_1.
  $[a,b]$:=$\mathsf{domain}(\nu)$
  $m = \frac{a+b}{2}$
  $\mathsf{zone}(\mathsf{child}_0(\nu)) = [b,m]$
  $\mathsf{zone}(\mathsf{child}_1(\nu)) = [m,a]$
  $\mathsf{domain}(\mathsf{child}_0(\nu))$, $\mathsf{domain}(\mathsf{child}_1(\nu))$ are formed as in (2.12)
  $\mathsf{domain}(\nu) = \mathsf{domain}(\mathsf{child}_0(\nu)) \cup \mathsf{domain}(\mathsf{child}_1(\nu))$
  $\mathsf{child}_0(\nu) := \nu_0$
  $\mathsf{child}_1(\nu) := \nu_1$
  $\mathrm{w}_0(\nu)$,$\mathrm{w}_1(\nu)$:= weights in (2.3) defined for $\mathsf{domain}(\nu_0)$, $\mathsf{domain}(\nu_1)$
else if ν is a leaf and $f(x)$ can be resolved by a Chebyshev $\mathsf{interpolant}$ with degree less than N **then**
  $\mathsf{interpolant}(\nu)$:=minimum degree $\mathsf{interpolant}$ $f(x)$ can be resolved by
    as determined by Chebfun
else
  refine($\mathsf{child}_0(\nu)$,f,N,t)
  refine($\mathsf{child}_1(\nu)$,f,N,t)
  $\mathsf{domain}(\nu) = \mathsf{domain}(\mathsf{child}_0(\nu)) \cup \mathsf{domain}(\mathsf{child}_1(\nu))$
  merge(ν,N)
end if

we formally describe how we refine our splitting; the merge method is described in section 2.2.1.

We first initialize the tree T with a single node ν where $\mathsf{domain}(\nu)$=$[a,b]$. Next we repeatedly call the refine method until each leaf of T has a Chebyshev interpolant that can resolve $f(x)$ with degree less than N, as seen in Algorithm 1. For a leaf ν, we determine if a Chebyshev interpolant can resolve $f(x)$ using Chebfun's standardChop method with $\mathsf{values}(\nu)$ (as described in Section 1). Using T we can evaluate $s_{[a,b]}(x)$ recursively as demonstrated in Algorithm 2.

As a simple example, we approximate the function $f(x) = \arctan\left(\frac{x-0.25}{0.001}\right)$ with $n_{\max} = 128$. In order to resolve to machine precision, a global Chebyshev interpolant on the interval $[-1,1]$ requires 25743 nodes while our method requires 412. Chebfun with non-overlapping splitting requires 381 nodes. Overlapping splittings will typically require more total nodes while offering the benefit of global smoothness. The result

Algorithm 2 v=eval(ν,x)

if ν is a leaf **then**
  p:=interpolant(ν)
  v:= $p(x)$
else
  v_0, v_1:=0
  w_0:=w$_0$(ν)
  w_1:=w$_1$(ν)
  for $k = 0, 1$ **do**
    if $x \in$ domain(child$_k$(ν)) **then**
      v_k:=eval(child$_k$(ν),x)
    end if
  end for
  v := $w_0(x)v_0 + w_1(x)v_1$
end if

can be seen in Figure 2.2.

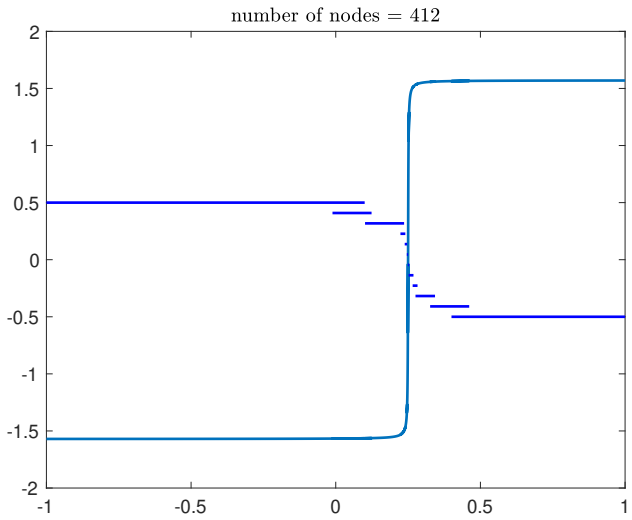


Figure 2.2: Plot of the partition of unity approximation with overlap parameter $t = 0.1$ for $f(x) = \arctan((x - 0.25)/0.001)$, where the solid blue lines represent the patches.

We can deduce from (2.6) that $s_{[a,b]}(x)$ will approximate $f(x)$. Moreover, our method implicitly creates a PU on the leaves of the tree through the product of the weights at each level.

Theorem 2.2.1. *Let an approximation $s_{[a,b]}(x)$ be as in (2.4). Then the tree that represents $s_{[a,b]}(x)$ implicitly defines a PU $\{w_k(x)\}_{k=1}^{M}$, where $w_k(x)$ has compact support*

over the kth leaf.

Proof. Suppose that on the domain $[a, b]$ we have PU's $\{w_{\ell k}(x)\}_{k=1}^{M_\ell}$, $\{w_{rk}(x)\}_{k=1}^{M_r}$ for the leaves of the left and right child respectively. We claim that

$$\{w_\ell(x)w_{\ell k}(x)\}_{k=1}^{M_\ell} \cup \{w_r(x)w_{rk}(x)\}_{k=1}^{M_r} \tag{2.14}$$

forms a PU over the leaves of the tree. We first observe that $w_\ell(x)w_{\ell k}(x)$ will have support in $\operatorname{supp}(w_{1k}(x))$, the domain of the respective leaf. This is similarly true for $w_r(x)w_{rk}(x)$.

Next suppose that $x \in \operatorname{sup}(w_\ell(x)) \cap \operatorname{sup}(w_r(x))^C$. Then $w_\ell(x) = 1$ and $w_r(x) = 0$, so

$$\sum_{k=1}^{M_\ell} w_\ell(x)w_{\ell k}(x) + \sum_{k=1}^{M_r} w_r(x)w_{rk}(x) = \sum_{k=1}^{M_\ell} w_{\ell k}(x) = 1, \tag{2.15}$$

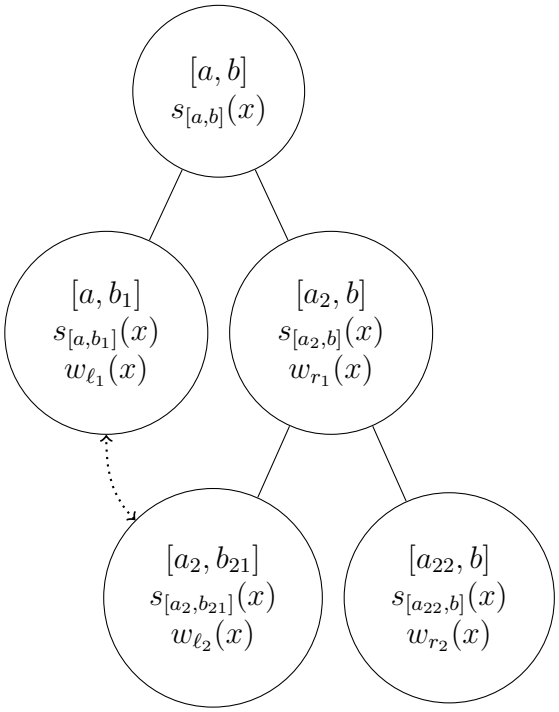
since $\{w_{\ell k}(x)\}_{k=1}^{M_\ell}$ is a PU. This is similarly true if $x \in \operatorname{sup}(w_\ell(x))^C \cap \operatorname{sup}(w_r(x))$. Finally if $x \in \operatorname{sup}(w_\ell(x)) \cap \operatorname{sup}(w_r(x))$ then

$$\begin{aligned}\sum_{k=1}^{M_\ell} w_\ell(x)w_{\ell k}(x) + \sum_{k=1}^{M_r} w_r(x)w_{rk}(x) &= w_\ell(x)\sum_{k=1}^{M_\ell} w_{\ell k}(x) + w_r(x)\sum_{k=1}^{M_r} w_{rk}(x) \\ &= w_\ell(x) + w_r(x) = 1.\end{aligned} \tag{2.16}$$

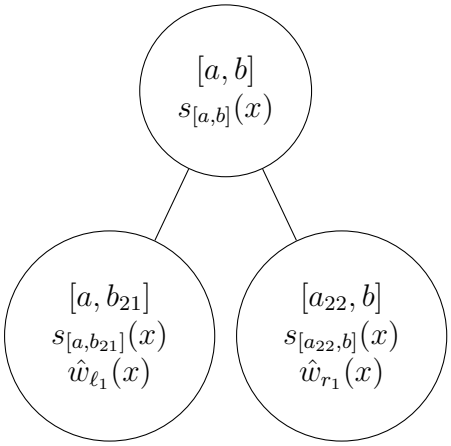
Thus by induction, we have that the product of weights through the binary tree for (2.4) implicitly creates a PU over the leaves. □

2.2.1 Merging

As we create the tree we opportunistically merge leaves for greater efficiency. If a particular location in the interval requires a great deal of refinement, the recursive splitting essentially performs a binary search for that location (as was noted about Chebfun splitting in [27]). The intermediate splits are not necessarily aiding with resolving the function; they are there just to keep the binary tree full. In Chebfun the recursive splitting phase is followed by a merging phase that discards counterproductive splits. We describe a similar merging operation here, but we allow these merges to take



(a) Tree before merging. Here $a_2 < b_1$ and $a_{22} < b_{21}$.



(b) Tree after merging.

Figure 2.3: An example of how leaves are merged, where each node is labeled with its domain, PU approximation and weight.

place whenever a leaf splits while its sibling does not, in order to keep the number of leaves from unnecessarily growing exponentially.

In Figure 2.3 we illustrate how we merge leaves; the interval $[a, b_1]$ is merged with $[a_2, b_{21}]$. Here we decide to merge if $f(x)$ can be resolved with an interpolant with degree less than N on the interval $[a, b_{21}]$. For the new tree we define the left weight $\hat{w}_{\ell_1}(x)$ in Figure 2.3 as

$$\hat{w}_{\ell_1}(x) = \begin{cases} 1 & x < a_{22}, \\ w_{\ell_2}(x) & \text{otherwise.} \end{cases} \tag{2.17}$$

Since $w_{\ell_2}(x) = 1$ for $x < a_{22}$, $\hat{w}_{\ell_1}(x)$ is smooth. For the right weight we use $\hat{w}_{r_1}(x) = w_{r_2}(x)$; these new weights form a PU. The PU approximation

$$\hat{s}(x) = w_{\ell 1}(x) s_{[a,b_1]}(x) + w_{r1}(x) s_{[a_2,b_{21}]}(x) \tag{2.18}$$

can be used to approximate $f(x)$ on $[a, b_{21}]$ since $f(x)$ is resolved at the leaves. In this case $s_{[a,b_{21}]}(x)$ is computed from sampling $\hat{s}(x)$. If the degree of $s_{[a,b_{21}]}(x)$ after Chebfun's chopping is less than N, we decide to merge. We explain in more detail the merging in Algorithm 3; here extend($w(x)$,$[a, b]$) piecewise extends the weight $w(x)$ in $[a, b]$ as in (2.17). We show the results for merging in Figure 2.4 with $f(x) = \frac{1}{x-1.001}$.

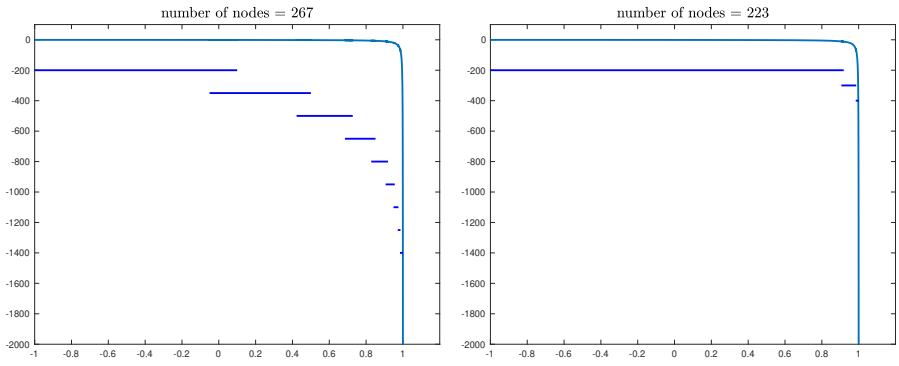
Algorithm 3 merge(ν,N)

if $\mathsf{child}_0(\nu)$ and $\mathsf{child}_0(\mathsf{child}_1(\nu))$ (child and grandchild of ν) are leaves and both of the intervals of the leaves can be resolved on **then**
  Define a new leaf ν_0
  $p_0(x)$:=$\mathsf{interpolant}(\mathsf{child}_0(\nu))$
  $p_1(x)$:=$\mathsf{interpolant}(\mathsf{child}_1(\mathsf{child}_0(\nu)))$
  $w_0(x)$:= $\mathrm{w}_0(\nu)$
  $w_1(x)$:= $\mathrm{w}_1(\nu)$
  $\hat{s}(x)$:=$w_0(x)p_0(x) + w_1(x)p_1(x)$
  if $\hat{s}(x)$ can be resolved by a Chebyshev $\mathsf{interpolant}$ $p(x)$ with degree less than N **then**
    $\mathsf{domain}(\nu_0)$:=$\mathsf{domain}(\mathsf{child}_0(\nu))\cup\mathsf{domain}(\mathsf{child}_0(\mathsf{child}_1(\nu)))$
    $\mathsf{interpolant}(\nu_0)$:=$p(x)$
    $\mathsf{points}(\nu_0)$:=Chebyshev grid of length $\deg(p(x))$ on $[a_0, b_1]$
    $\hat{w}_0(x)$:= $\mathrm{w}_0(\mathsf{child}_1(\nu))$
    $\hat{w}_1(x)$:= $\mathrm{w}_1(\mathsf{child}_1(\nu))$
    $\mathrm{w}_0(\nu)$:= extend($\hat{w}_0(x)$,$\mathsf{domain}(\nu_0)$)
    $\mathrm{w}_1(\nu)$:= $\hat{w}_1(x)$
    $\mathsf{child}_0(\nu)$:= ν_0
    $\mathsf{child}_1(\nu)$:= $\mathsf{child}_1(\mathsf{child}_1(\nu))$
  end if
else if $\mathsf{child}_1(\nu)$ is a leaf and $\mathsf{child}_1(\mathsf{child}_0(\nu))$ is a leaf (and exists) **then**
  inv(merge(ν)) (i.e. apply the algorithm, except swap 0 and 1)
end if

2.2.2 Differentiation matrices

Next we demonstrate how to construct a first derivative matrix; higher derivative matrices can be similarly constructed. Suppose we have constructed a splitting represented with the tree T. For each node ν of the tree, we add the following methods:

- $\mathsf{points}(\nu)$:= provides the Chebyshev points of the leaves of ν
- $\mathsf{leafpoints}(\nu)$:= provides the Chebyshev points of T in $\mathsf{domain}(\nu)$ i.e. $\mathsf{points}(\mathrm{root}(T)) \cap \mathsf{domain}(\nu)$



(a) Tree before merging. (b) Tree after merging.

Figure 2.4: An example of how the PUM with $t = 0.08$, $N = 128$ resolves $f(x) = \frac{1}{x-1.0005}$ without merging (a) and after merging (b).

- $\mathsf{pointindex}(\nu)$:=gives the index of $\mathsf{points}(\nu)$ with respect to the points of the parent of ν (if ν is a child)
- $\mathsf{leafpointindex}(\nu)$:=gives the index of $\mathsf{leafpoints}(\nu)$ with respect to the leafpoints of the parent of ν (if ν is a child).

Let $[\alpha, \beta] = \mathsf{domain}(\nu)$. We want to construct matrices M, D such that

$$\begin{aligned} M\ f(x)|_{\mathsf{points}(\nu)} &= s_{[\alpha,\beta]}(x)\big|_{\mathsf{leafpoints}(\nu)}\,, \\ D\ f(x)|_{\mathsf{points}(\nu)} &= \frac{d}{dx} s_{[\alpha,\beta]}(x)\Big|_{\mathsf{leafpoints}(\nu)}\,. \end{aligned} \tag{2.19}$$

Let $I_k = \mathsf{domain}(\mathsf{child}_k(\nu))$, $w_k(x) = \mathrm{w_k}(\nu)$ for $k = 0, 1$. Then

$$\begin{aligned} s_{[\alpha,\beta]}(x)\big|_{\mathsf{leafpoints}(\nu)} &= \sum_{k=0}^{1} w_k(x) s_{I_k}(x)|_{\mathsf{leafpoints}(\nu)}\,, \\ \frac{d}{dx} s_{[\alpha,\beta]}(x)\Big|_{\mathsf{leafpoints}(\nu)} &= \sum_{k=0}^{1} \left(w_k(x) \frac{d}{dx} s_{I_k}(x) + \frac{d}{dx} w_k(x) s_{I_k}(x) \right)\Big|_{\mathsf{leafpoints}(\nu)}\,. \end{aligned} \tag{2.20}$$

Thus we can recursively build up the differentiation matrix through the tree T. Due to the support of the weights, for each term in (2.20) we only need evaluate the approximation $s_{I_k}(x)$ (or its derivative) for $\mathsf{leafpoints}(\nu) \cap I_k$, i.e. $\mathsf{leafpoints}(\mathsf{child}_k(\nu))$. We describe how to construct the differentiation recursively in Algorithm 4, using MATLAB notation for matrices. At each leaf the interpolation matrix M has entries given

Algorithm 4 $[M, D]$=diffmatrix(ν)

if ν is a leaf **then**
  M:= the Chebyshev barycentric matrix from $\mathsf{points}(\nu)$ to $\mathsf{leafpoints}(\nu)$
  D_x:= Chebyshev differentiation matrix with grid $\mathsf{points}(\nu)$.
  D:=MD_x.
else
  M, D:=zeros(length($\mathsf{leafpoints}(\nu)$),length($\mathsf{points}(\nu)$))
  for $k = 0, 1$ **do**
    $[M_k,D_k]$:= diffmatrix($\mathsf{child}_k(\nu)$)
    $M(\mathsf{leafpointindex}(\mathsf{child}_k(\nu)),\mathsf{pointindex}(\mathsf{child}_k(\nu))) =$
      $\hookrightarrow$ diag $\left(w_k|_{\mathsf{leafpoints}(\mathsf{child}_k(\nu))}\right)$*$M_k$;
    $D(\mathsf{leafpointindex}(\mathsf{child}_k(\nu)),\mathsf{pointindex}(\mathsf{child}_k(\nu))) =$
      $\hookrightarrow$ diag $\left(w_k|_{\mathsf{leafpoints}(\mathsf{child}_k(\nu))}\right)$*$D_k$+
      $\hookrightarrow$ diag $\left(\frac{d}{dx}\, w_k|_{\mathsf{leafpoints}(\mathsf{child}_k(\nu))}\right)$*$M_k$;
  end for
end if

by the barycentric interpolation formula based on second-kind Chebyshev points, as produced by the Chebfun command `barymat` [26].

For $x \in [\alpha, \beta]$ we only need to evaluate the local approximations for the patches x belongs to; this implies that the differentiation matrices will be inherently sparse. For example, Figure 2.5 shows the sparsity of the first derivative matrix for the tree generated in Figure 2.2. In this case, we have a sparsity ratio of around 76%.

2.3 PUM for boundary-value problems

Our method can be applied to solve linear and nonlinear boundary-value problems. For instance, consider a simple Poisson problem with zero boundary conditions:

$$\begin{aligned} u''(x) &= f(x) \text{ for } -1 < x < 1 \\ u(-1) &= 0, u(1) = 0. \end{aligned} \tag{2.21}$$

Suppose that we have differentiation and interpolation matrices D_{xx} and M from section 2.2.2, X is the set of Chebyshev points over all the leaves, and that X_I, X_B are the respective interior and boundary points of X. Let E_I and E_B be the matrices that

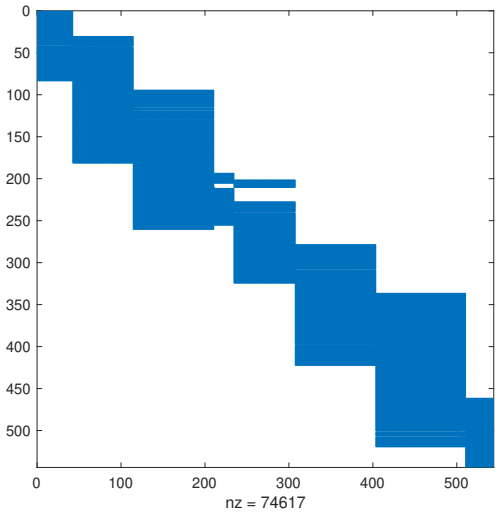


Figure 2.5: Sparsity of the first derivative matrix for the tree generated for Figure 2.2.

map a vector to its subvector for the interior and boundary indices respectively. Let F be the vector of values used for the local interpolants (i.e. if we had only two leaves whose interpolants used values F_1, F_2, we set $F = [F_1^T F_2^T]^T$). In order to find a PUM approximation $s(x)$ that approximates (2.21) we find F by solving the following linear system:

$$\begin{bmatrix} E_I D_{xx} \\ E_B M \end{bmatrix} \begin{bmatrix} E_I F \\ E_B F \end{bmatrix} = \begin{bmatrix} f|_{X_I} \\ 0 \end{bmatrix}. \tag{2.22}$$

Algorithm 5 builds an adaptive solution for the BVP. We first construct a PU approximation $s(x)$ by solving the discretized system in (2.22). Sampling with $s(x)$, we use Algorithm 1 to determine if the solution is refined and split leaves that are determined to be unrefined. Here we also allow merging for a node with resolved left and right leaves (i.e., the left and right leaves can be merged back together).

Algorithm 5 T=refineBVP(N,t,BVP)

Define T as a tree with a single node with the domain of the BVP.
while T has unrefined leaves **do**
  Find values for the interpolants F of the leaves of T by solving a discretized system defined by the interpolation and differentiation matrices of T.
  sample(T,F)
  $s(x) = \text{eval}(\text{root}(T), x)$ (the PU approximation)
  sample(T,$s(x)$)
  refine(root(T),N,t)
end while

2.3.1 BVP example

We solve the stationary Burgers equation on the interval $[0, 1]$ with Robin boundary conditions [59]:

$$\begin{aligned} \nu u''(x) - u(x)u'(x) &= 0 \\ \nu u'(0) - \kappa(u(0) - \alpha) &= 0 \\ \nu u'(1) + \kappa(u(1) + \alpha) &= 0 \end{aligned} \tag{2.23}$$

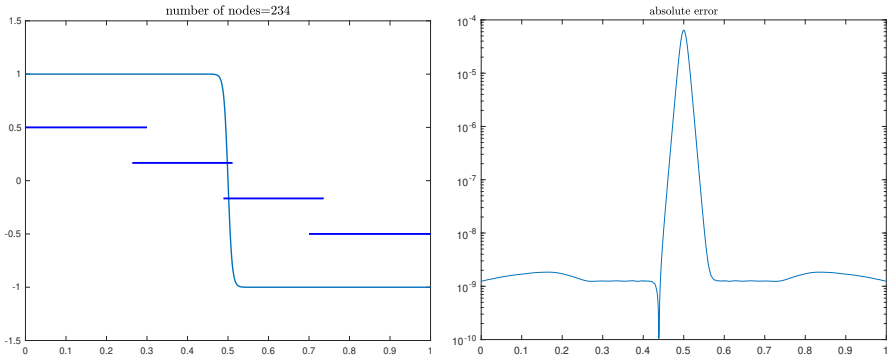
which has nontrivial solution

$$u(x) = -\beta \tanh\left(\frac{1}{2}\beta\nu^{-1}\left(x - \frac{1}{2}\right)\right) \tag{2.24}$$

where β satisfies

$$-\frac{1}{2}\beta^2 \text{sech}^2\left(\frac{1}{4}\beta\nu^{-1}\right) + \kappa\left[\alpha - \beta\tanh\left(\frac{1}{4}\beta\nu^{-1}\right)\right] = 0. \tag{2.25}$$

We choose $\nu = 5 \times 10^{-3}, \alpha = 1$, and $\kappa = 2$. We use `fsolve` in MATLAB to solve the BVP, supplying the Jacobian of the discretized nonlinear system. Starting with a linear guess $u(x) = 0$, we update the solution from the latest solve (i.e. if the solution $s(x)$ from Algorithm 5 is determined to be unresolved, we use it as the next initial guess). For this problem we set the Chebfun chopping tolerance to 10^{-10}. Our solution was resolved to the tolerance we set after four nonlinear solves; as seen in Figure 2.6, the final approximation had 234 nodes and the absolute error was less than 10^{-4} as seen in Figure 2.6. On a machine with processor 2.6 GHz Intel Core i5, the solution was found in 1.3 seconds.



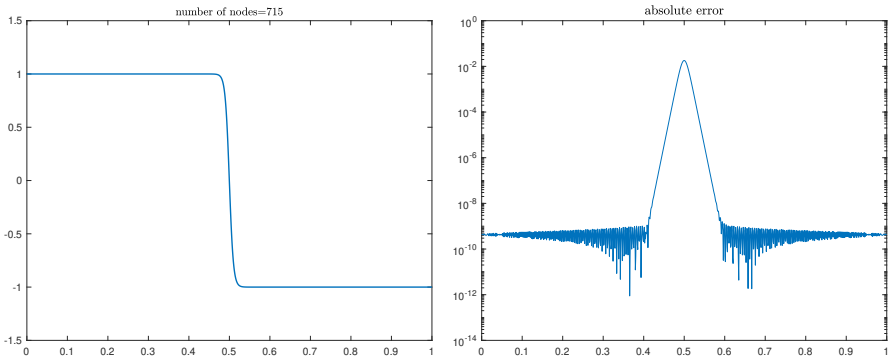
(a) Solution with subintervals plotted. (b) Plot of the error.

Figure 2.6: Numerical solution using the PU method and residual for the BVP (2.23) with $\nu = 5 \times 10^{-3}$, $t = 0.1$, and $N = 128$.

We preformed a similar experiment but instead used global Chebyshev interpolants. We adapt by increasing the degree of the polynomial from n to floor$(1.5n)$, starting with $n = 128$. We stop when we have a solution that is refined to the tolerance 10^{-10} (same as before). Both the solution and residual are in Figure 2.7; here we have the absolute error is higher at 1.8e-2. The solution took 3.2 minutes on the same machine. There are two main reasons why the global solution performs much slower. First, in order to resolve the true solution with the tolerance 10^{-10}, the global Chebyshev solution requires 766 nodes versus 234 for the PU approximation. Secondly, when adapting with the PUM, if a leaf is determined to be refined, the number of nodes is reduced as dictated in Algorithm 1 and the leaf is not split in further iterations. This keeps the total number of nodes lower while adapting.

2.4 Discussion

Our method offers a simple way to adaptively construct infinitely smooth approximations of functions that are given explicitly or that solve BVPs. By recursively constructing the PU weights with the binary tree, we avoid the need to determine the neighbors of each patch (as would be needed with the standard Shepard's PU weights). While this is not a serious issue in one dimension, the complexity of how the patches overlap increases with higher dimension.



(a) Numerical solution. (b) Plot of the error.

Figure 2.7: Numerical solution using the global Chebyshev method and residual for the BVP (2.23) with $\nu = 5 \times 10^{-3}$.

Our method leaves room for improvement. For instance, while merging helps reduce the number of nodes, in cases where we have a singularity right above the split the PU method over-resolves in the overlap; this can be seen in Figure 2.2. The source of the problem is that patches may be adjacent in space but not in the tree. This could be resolved by a more robust merging algorithm. Alternatively we could determine an optimal splitting location through a Chebyshev-Padé approximation as in [27], but the PU adds a layer of complexity since we must optimize not just for the splitting location but the size of the overlap.

Additionally it is possible to construct weights that are not C^∞ but have smaller norms in their derivatives. For instance,

$$w_\ell(x) = \begin{cases} 1 & x \le -t \\ \frac{1}{4t^3}x^3 - \frac{3}{4t}x + \frac{1}{2} & -t \le x \le t \\ 0 & x > t \end{cases} \qquad (2.26)$$
$$w_r(x) = 1 - w_\ell(x)$$

defines a $C^1[-1,1]$ piecewise cubic partition of unity, where $\|w_\ell'(x)\|_\infty = \frac{3}{4t}$. If a BVP requires higher smoothness, we could similarly construct a higher degree polynomial for the weights.

Chapter 3

AN ADAPTIVE PARTITION OF UNITY METHOD FOR MULTIVARIATE CHEBYSHEV POLYNOMIAL APPROXIMATIONS

3.1 Introduction

One aspect of the low-rank approximations used by Chebfun2 and Chebfun3 is that they are highly anisotropic. That is, rotation of the coordinate axes can transform a rank-one or low-rank function into one with a much higher rank, greatly increasing the time required for function construction and manipulations. This issue is considered in detail in [71].

In this chapter we propose decomposing a hyperrectangular domain by adaptive, recursive bisections in one dimension at a time, generalizing earlier work in one dimension [3]. The resulting subdomains are defined to be overlapping, and on each we employ simple tensor-product Chebyshev polynomial interpolants. In order to define a global smooth approximation, we use a partition of unity to blend together the subdomains. This allows the approximation to capture highly localized function features while remaining computationally tractable.

Our use of an adaptive decomposition allows us to approximate on such domains with great flexibility. If a base subdomain is hyperrectangular, we proceed with a tensor-product interpolation for speed, but if its intersection with the global domain is nonrectangular, we can opt for a different representation. We need not be concerned with having a very large number of degrees of freedom in any local subproblem, since further subdivision is available, so the local algorithm need not be overly sophisticated.

The adaptive construction of function approximations is based on binary trees, as explained in section 3.2. In section 3.3 we describe fast algorithms for evaluation,

arithmetic combination, differentiation, and integration of the resulting tree-based approximations. Numerical experiments over hyperrectangles in section 3.4 demonstrate that the tree-based approximations exhibit far less anisotropy than do Chebfun2 and Chebfun3. Our implementation is faster than Chebfun2 and Chebfun3 on all tested examples—sometimes by orders of magnitude—except for examples of very low rank, for which all the methods are acceptably fast. In section 3.5 we describe and demonstrate approximation on nonrectangular domains using a simple linear least-squares approximation by the tensor-product Chebyshev basis. While these results are preliminary, we think they show enough promise to merit further investigation.

3.2 Adaptive construction

Let $\Omega = \{\mathbf{x} \in \mathbb{R}^d : x_i \in [a_i, b_i], i = 1, \ldots, d\}$ be a hyperrectangle, and suppose we wish to approximate $f : \Omega \to \mathbb{R}$. Our strategy is to cover Ω with overlapping subdomains, on each of which f is well-approximated by a multivariate polynomial, and use a partition of unity to construct a global approximation. We defer a description of the partition of unity scheme to section 3.3. In this section we describe an adaptive procedure for obtaining the overlapping domains and individual approximations over them.

The domains are constructed from recursive bisections of Ω into nonoverlapping hyperrectangular *zones*. Given a zone $\prod_{j=1}^{d}[\alpha_j, \beta_j]$, we extend it to a larger domain $\prod_{j=1}^{d}[\bar{\alpha}_j, \bar{\beta}_j]$ by fixing a parameter $t > 0$, defining

$$\delta_j = \frac{\beta_j - \alpha_j}{2}(1 + t), \quad j = 1, \ldots, d, \tag{3.1}$$

and then setting

$$\bar{\alpha}_j = \max\{a_j, \beta_j - \delta_j\}, \quad \bar{\beta}_j = \min\{\alpha_j + \delta_j, b_j\}. \tag{3.2}$$

In words, the zone is extended on all sides by an amount proportional to its width in each dimension, up to the boundary of the global domain Ω.

We define a binary tree $\mathcal{T}$ with each node ν having the following properties:

- $\mathsf{zone}(\nu)$: zone associated with ν

- $\mathsf{domain}(\nu)$: domain associated with ν
- $\mathsf{isdone}(\nu)$: n-vector of boolean values, where isdone_j indicates whether the domain is determined to be sufficiently resolved in the jth dimension
- $\mathsf{child}_0(\nu)$,$\mathsf{child}_1(\nu)$: left and right subtrees of ν (empty for a leaf)
- $\mathsf{splitdim}(\nu)$: the dimension in which ν is split (empty for a leaf)

A leaf node has the following additional properties:

- $\mathsf{grid}(\nu)$: tensor-product grid of Chebyshev 2nd-kind points mapped to $\mathsf{domain}(\nu)$
- $\mathsf{values}(\nu)$: function values at $\mathsf{grid}(\nu)$
- $\mathsf{interpolant}(\nu)$: polynomial interpolant of $\mathsf{values}(\nu)$ on $\mathsf{grid}(\nu)$

If ν is a leaf, its domain is constructed by extending $\mathsf{zone}(\nu)$ as in (3.2). Otherwise, $\mathsf{domain}(\nu)$ is the smallest hyperrectangle containing the domains of its children.

Let f be the scalar-valued function on Ω that we wish to approximate. A key task is to compute, for a given leaf node ν, the polynomial $\mathsf{interpolant}(\nu)$, and determine whether f is sufficiently well approximated on $\mathsf{domain}(\nu)$ by it. First we sample f at a Chebyshev grid of size N^d on $\mathsf{domain}(\nu)$. This leads to the interpolating polynomial

$$\tilde{p}(\mathbf{x}) = \sum_{i_1=0}^{N-1} \cdots \sum_{i_d=0}^{N-1} C_{i_1,\ldots,i_d} T_{i_1}(x_1) \cdots T_{i_d}(x_d), \tag{3.3}$$

where the coefficient array C can be computed by FFT in $\mathcal{O}(N^d \log N)$ time [51]. Following the practice of Chebfun3t [37], for each $j = 1, \ldots, d$, we define a scalar sequence $\gamma^{(j)}$ by summing $|C_{i_1,\ldots,i_d}|$ over all dimensions except the jth. To each of these sequences we apply Chebfun's `StandardChop` algorithm, which attempts to measure decay in the coefficients in a suitably robust sense [5]. Let the output of `StandardChop` for sequence $\gamma^{(j)}$ be n_j; this is the degree that `StandardChop` deems to be sufficient for resolution at a user-set tolerance. If $n_j < N$ we say that the function is resolved in dimension j on ν. If f is resolved in all dimensions on ν, then we truncate the interpolant sums in (3.3) at the degrees n_j and store the samples of f on the corresponding smaller tensor-product grid.

Algorithm 6 refine(ν,f,N,t)

if ν is a leaf **then**
  Sample f on $\mathsf{grid}(\nu)$
  Determine chopping degrees $n_1, \ldots, n_d$
  for each j with $\mathsf{isdone}(\nu)_j = \text{FALSE}$ **do**
    if $n_j < N$ **then**
      $\mathsf{isdone}(\nu)_j := \text{TRUE}$
    else
      split(ν,j,t)
    end if
  end for
  if all $\mathsf{isdone}(\nu)$ are TRUE **then**
    Truncate (3.3) at degrees $n_1, \ldots, n_d$ to define $\mathsf{grid}(\nu)$, $\mathsf{values}(\nu)$, $\mathsf{interpolant}(\nu)$
  else
    refine(ν,f,N,t)
  end if
else
  refine($\mathsf{child}_0(\nu)$,f,N,t)
  refine($\mathsf{child}_1(\nu)$,f,N,t)
end if

Algorithm 6 describes a recursive adaptation procedure for building the binary tree $\mathcal{T}$, beginning with a root node whose zone and domain are both the original hyperrectangle Ω. For a non-leaf input, the algorithm is simply called recursively on the children. For an input node that is currently a leaf of the tree, the function f is sampled, and chopping is used in each unfinished dimension to determine whether sufficient resolution has been achieved. Each dimension that is deemed to be resolved is marked as finished. If all dimensions are found to be finished, then the interpolant is chopped to the minimum necessary length in each dimension, and the node will remain a leaf. Otherwise, the node is split in all unfinished dimensions using Algorithm 7, and Algorithm 6 is applied recursively. Note that the descendants of a splitting inherit the isdone property that marks which dimensions have been finished, so no future splits are possible in such dimensions within this branch.

Algorithm 7 split(ν,j,t)

if ν is a leaf **then**
  $\mathsf{splitdim}(\nu)=j$
  Define new nodes ν_0, ν_1
  $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$ be the subintervals from $\mathsf{zone}(\nu)$
  Let $m := \frac{b_j+a_j}{2}$
  Let $\mathsf{zone}(\nu_0) := [a_1, b_1] \times \cdots \times [a_{j-1}, b_{j-1}] \times [a_j, m] \times [a_{j+1}, b_{j+1}] \times \cdots \times [a_d, b_d]$
  Let $\mathsf{zone}(\nu_1) := [a_1, b_1] \times \cdots \times [a_{j-1}, b_{j-1}] \times [m, b_j] \times [a_{j+1}, b_{j+1}] \times \cdots \times [a_d, b_d]$
  for $k = 0, 1$ **do**
    Define $\mathsf{domain}(\nu_k)$ from $\mathsf{zone}(\nu_k)$ with parameter t as in (3.2)
    Define $\mathsf{grid}(\nu_k)$ as Chebyshev tensor-product grid of size N^d in $\mathsf{domain}(\nu_k)$
    Let $\mathsf{isdone}(\nu_k)$:= $\mathsf{isdone}(\nu)$
  end for
else
  split($\mathsf{child}_0(\nu)$,k,t)
  split($\mathsf{child}_1(\nu)$,k,t)
end if

3.3 Computations with the tree representation

The procedure of the preceding section constructs a binary tree $\mathcal{T}$ whose leaves each hold an accurate representation of f over a subdomain. These subdomains overlap, and constructing a global partition of unity approximation from them is straightforward.

Define the C^∞ function

$$\psi_0(x) = \begin{cases} \exp\left(1 - \frac{1}{1-x^2}\right) & |x| \le 1, \\ 0 & |x| > 1, \end{cases} \tag{3.4}$$

and let

$$\ell(x; a, b) = 2\frac{x-a}{b-a} - 1 \tag{3.5}$$

be the affine map from $[a, b]$ to $[-1, 1]$. Suppose ν is a leaf of $\mathcal{T}$ with domain $\Omega_\nu = \prod[\bar{\alpha}_j, \bar{\beta}_j]$. Then we can define the smoothed-indicator or bump function

$$\psi_\nu(\mathbf{x}) = \prod_{j=1}^{d} \psi_0\big(\ell(x_j; \bar{\alpha}_j, \bar{\beta}_j)\big). \tag{3.6}$$

Next we use Shepard's method [73] to define a partition of unity $\{w_\nu(\mathbf{x})\}$, indexed by the leaves of $\mathcal{T}$:

$$w_\nu(\mathbf{x}) = \frac{\psi_\nu(\mathbf{x})}{\sum_{\mu \in \text{leaves}(\mathcal{T})} \psi_\mu(\mathbf{x})}. \tag{3.7}$$

We have $\sum_{\nu \in \text{leaves}(\mathcal{T})} w_\nu(\mathbf{x}) = 1$, which makes $\{w_\nu(\mathbf{x})\}$ a partition of unity. This implies that $w_\nu(\mathbf{x}) = 1$ for any $\mathbf{x}$ that lies in ν and no other patches. Thus if we assume that weight functions are supported only in their respective domains, smoothness of the partition of unity functions requires overlap between neighboring patches.

Let s_ν be the polynomial interpolant of f over the domain of node ν. Then the global partition of unity approximant is

$$s(\mathbf{x}) = \sum_{\nu \in \text{leaves}(\mathcal{T})} w_\nu(\mathbf{x}) s_\nu(\mathbf{x}). \tag{3.8}$$

Despite consisting of separate local approximations from a partitioned domain, the global approximation (3.8) remains infinitely smooth while avoiding explicit global matching constraints. This permits rapid (in principle, beyond all orders) convergence to smooth functions, as well as generating continuous derivative approximations [73].

While this approximation is globally continuous it is still local in some sense. As an example, in Figure 3.1 we plot the overlapping patches on the domain of a patch ν for the partition of unity approximation of $\arctan(3(y^2 + x))$ (which can be seen in Figure 3.3). We see that in the interior of the patch that the approximation (3.8) would consist only of the polynomial approximation $s_\nu(\mathbf{x})$, and in the overlap would blend neighboring approximations with the partition of unity.

Next we describe efficient algorithms using the tree representation of the global approximant to perform common numerical operations such as evaluation at points, basic binary arithmetic operations on functions, differentiation, and integration.

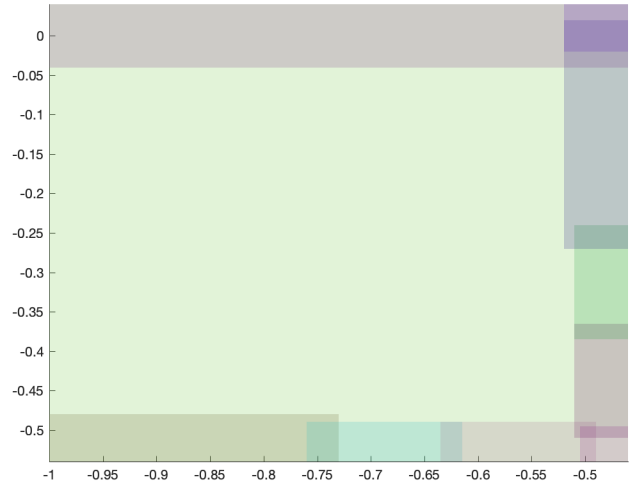


Figure 3.1: Plot of the subdomains formed from the partition of unity method for $\arctan(3(y^2 + x))$ on a local patch with domain $[-1, -0.46] \times [-0.54, 0.04]$.

3.3.1 Evaluation

Note that (3.7)– (3.8) can be rearranged into

$$s(\mathbf{x}) = \sum_{\nu \in \text{leaves}(\mathcal{T})} \frac{s_\nu(\mathbf{x})\psi_\nu(\mathbf{x})}{\sum_{\mu \in \text{leaves}(\mathcal{T})} \psi_\mu(\mathbf{x})} = \frac{\sum_{\nu \in \text{leaves}(\mathcal{T})} s_\nu(\mathbf{x})\psi_\nu(\mathbf{x})}{\sum_{\mu \in \text{leaves}(\mathcal{T})} \psi_\mu(\mathbf{x})}. \tag{3.9}$$

This formula suggests a recursive approach to evaluating the numerator and denominator, presented in Algorithm 8. Using it, only leaves containing $\mathbf{x}$ and their ancestors are ever visited. A similar approach was described in [65].

Algorithm 8 can easily be vectorized to evaluate $s(\mathbf{x})$ at multiple points, by recursively calling each leaf with all values of $\mathbf{x}$ that lie within its domain. In the particular case when the evaluation is to be done at all points in a Cartesian grid, it is worth noting that the leaf-level interpolant in (3.3) can be evaluated by a process that yields significant speedup over a naive approach. As a notationally streamlined example, say that the desired values of $\mathbf{x}$ are $(\xi_{j_1}, \ldots, \xi_{j_d})$, where each j_k is drawn from $\{1, \ldots, M\}$, and that the array of polynomial coefficients is of full size $O(N^d)$.

Algorithm 8 $[S,P]$=numden(ν,$\mathbf{x}$)

$S = 0$, $P = 0$
if ν is a leaf **then**
  $S = \psi_\nu(\mathbf{x})$
  $P = S \cdot \mathsf{interpolant}(\nu)(\mathbf{x})$
else
  for $k = 0, 1$ **do**
    if $\mathbf{x} \in \mathsf{domain}(\mathsf{child}_k(\nu))$ **then**
      $[S_k, P_k]$ = numden($\mathsf{child}_k(\nu)$,$\mathbf{x}$)
      $S = S + S_k$
      $P = P + P_k$
    end if
  end for
end if

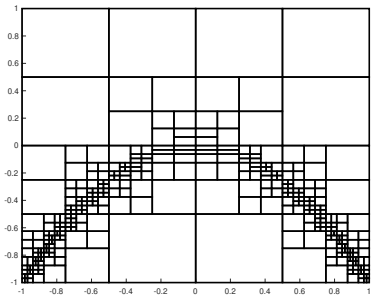
Express (3.3) as

$$\sum_{i_1=0}^{N-1} \cdots \sum_{i_d=0}^{N-1} C_{i_1,\ldots,i_d} T_{i_1}(\xi_{j_1}) \cdots T_{i_d}(\xi_{j_d}) = \sum_{i_1=0}^{N-1} T_{i_1}(\xi_{j_1}) \sum_{i_2=0}^{N-1} T_{i_2}(\xi_{j_2}) \cdots \sum_{i_d=0}^{N-1} C_{i_1,\ldots,i_d} T_{i_d}(\xi_{j_d}). \quad (3.10)$$

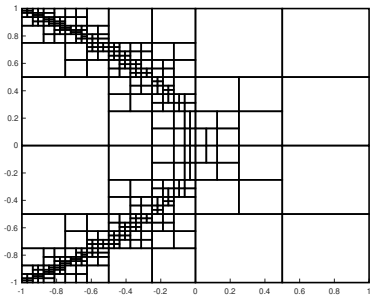
The innermost sum yields $N^{d-1}M$ unique values, each taking $\mathcal{O}(N)$ time to compute. At the next level there are $N^{d-2}M^2$ values, and so on, finally leading to the computation of all M^d interpolant values. This takes $\mathcal{O}(MN(M+N)^{d-1})$ operations, as opposed to $\mathcal{O}(M^d N^d)$ when done naively.

3.3.2 Binary arithmetic operations

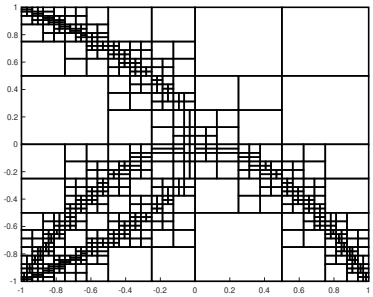
Suppose we have two approximations $s_1(\mathbf{x})$, $s_2(\mathbf{x})$, represented by trees $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively, and we want to construct a tree approximation for $s_1 \circ s_2$, where $\circ$ is one of the operators $+$, $-$, $\times$, or $\div$. If $\mathcal{T}_1$ and $\mathcal{T}_2$ have identical tree structures, then it is straightforward to operate leafwise on the polynomial approximations. In the cases of multiplication and division, the resulting tree may have to be refined further using Algorithm 7, since these operations typically result in polynomials of degree greater than the operands.



(a) Zone plot of $f_1(x, y)$



(b) Zone plot of $f_2(x, y)$



(c) Zone plot of $f_1(x, y) + f_2(x, y)$

Figure 3.2: Zone plots for $f_1(x, y)$,$f_2(x, y)$ and $f_1(x, y) + f_2(x, y)$.

If the trees $\mathcal{T}_1$ and $\mathcal{T}_2$ are not structurally identical, we are free to use Algorithm 7 to construct an approximation by sampling values of $s_1 \circ s_2$. However, the tree of $s_1 \circ s_2$ likely shares refinement structure with both $\mathcal{T}_1$ and $\mathcal{T}_2$. For example, Figure 3.2 shows the refined zones of the trees for $\arctan(100(x^2 + y))$, $\arctan(100(x + y^2))$, and their sum. Thus in practice we merge the trees $\mathcal{T}_1$ and $\mathcal{T}_2$ using Algorithm 14, presented in Appendix A. The merged tree, whose leaves contain sampled values of the result, may then be refined further if chopping tests then reveal that the result is not fully resolved.

3.3.3 Differentiation

Differentiation of the global approximant (3.8) results in two groups of terms:

$$\frac{\partial}{\partial x_j} s(\mathbf{x}) = \sum_{\nu \in \text{leaves}(\mathcal{T})} w_\nu(\mathbf{x}) \frac{\partial}{\partial x_j} s_\nu(\mathbf{x}) + \sum_{\nu \in \text{leaves}(\mathcal{T})} s_\nu(\mathbf{x}) \frac{\partial}{\partial x_j} w_\nu(\mathbf{x}).$$

The first sum is a partition of unity approximation of leafwise differentiated interpolants. That is, we simply apply standard spectral differentiation to the data stored in the leaves of $\mathcal{T}$. Although it may seem surprising at first, we can define the desired derivative approximation solely in terms of this first sum, and neglect the second with little penalty.

Theorem 3.3.1. *Define*

$$s^{(j)}(\mathbf{x}) = \sum_{\nu \in leaves(\mathcal{T})} w_\nu(\mathbf{x}) \frac{\partial}{\partial x_j} s_\nu(\mathbf{x}). \tag{3.11}$$

Then for all $\mathbf{x} \in \Omega$,

$$\left| s^{(j)}(\mathbf{x}) - \frac{\partial f}{\partial x_j}(\mathbf{x}) \right| \leq \sum_{\mathbf{x} \in \mathit{domain}(\nu)} w_\nu(\mathbf{x}) \left| \frac{\partial s_\nu}{\partial x_j}(\mathbf{x}) - \frac{\partial f}{\partial x_j}(\mathbf{x}) \right|. \tag{3.12}$$

Proof. By the partition of unity property,

$$s^{(j)}(\mathbf{x}) - \frac{\partial f}{\partial x_j}(\mathbf{x}) = \sum_{\nu \in \mathrm{leaves}(\mathcal{T})} w_\nu(\mathbf{x}) \left[\frac{\partial}{\partial x_j} s_\nu(\mathbf{x}) - \frac{\partial f}{\partial x_j}(\mathbf{x}) \right].$$

The result follows because $w_\nu(\mathbf{x}) = 0$ if $\mathbf{x} \notin \mathsf{domain}(\nu)$. □

Hence if $\mathbf{x}$ is not in an overlap region, the error in the global derivative approximation $s^{(j)}$ is the same as for the local approximant. Otherwise, it is bounded—pessimistically, since the weights are positive and sum to unity pointwise—by the sum of errors in all the contributing approximants. Since no point can be in more than 2^d subdomains (and then only near a meeting of hyperrectangle corners), we feel this error is acceptable in two and three dimensions.

3.3.4 Integration

The simplest and seemingly most efficient approach to integrating over the domain is to do so piecewise over the nonoverlapping zones,

$$\int_\Omega f(\mathbf{x}) d\mathbf{x} = \sum_{\nu \in \mathrm{leaves}(\mathcal{T})} \int_{\mathsf{zone}(\nu)} f(\mathbf{x}) d\mathbf{x}. \tag{3.13}$$

Since the leaf interpolants are defined natively over the overlapping domains, they must be resampled at Chebyshev grids on the zones, after which Clenshaw-Curtis quadrature is applied.

3.4 Numerical experiments

All the following experiments were performed on a computer with a 2.6 GHz Intel Core i5 processor in version 2017a of MATLAB. Our code, which uses a serial object-oriented recursive implementation of the algorithms, is available for download.[1] Comparisons to Chebfun2 and Chebfun3 were done using Chebfun version 5.5.0. We also tried to use the Sparse Grid Interpolation Toolbox [44], but on all the examples we were unable to get it close to our desired error tolerances within its hard-coded limits on sparse grid depth.

3.4.1 2D experiments

We first test the 2D functions $\log(1 + (x^2 + y^4)/10^{-5})$, $\arctan((x + y^2)/10^{-2})$, $\frac{10^{-4}}{(10^{-4}+x^2)(10^{-4}+y^2)}$, Franke's function [30], the smooth functions from the Genz family test package [33], and the "peg" examples from [71]. For each function we record the time of construction, the time to evaluate on a 200×200 grid, and the max observed error on this grid. Table 3.1 shows the results for the new method. For the low-rank test cases, the methods are comparable, with neither showing a consistent advantage; most importantly, both methods are fast enough for interactive computing. In the tests of higher-rank functions, the tree-based method exhibits a clear, sometimes dramatic, advantage in construction time. Moreover, the tree method remains fast enough for interactive computing even as the total number of nodes exceeds 1.6 million. We present plots of the functions and adaptively generated subdomains for the first three test functions in Figures 3.3-3.4.

One important aspect of low-rank approximation is that it is inherently non-isotropic. Consider the 2D "plane wave bump"

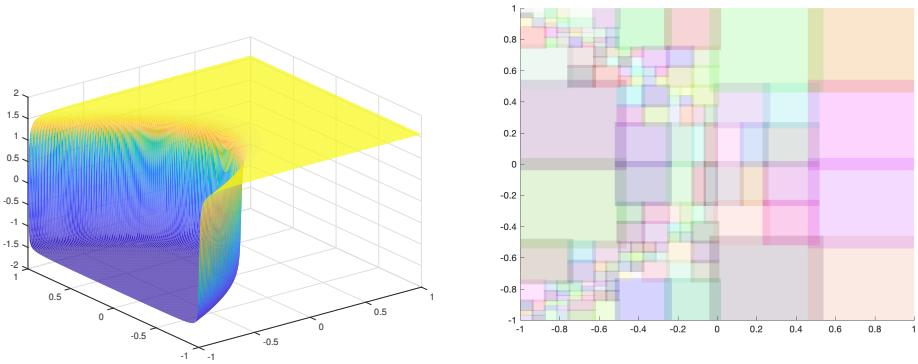
$$f(x, y) = \arctan(250(\cos(t)x + \sin(t)y)) \tag{3.14}$$

whose normal makes an angle t with the positive x-axis. We compare the construction times of our method to Chebfun2 for $t \in [0, \pi/4]$ in Figure 3.5. We observe the

[1] https://github.com/kevinwaiton/PUchebfun

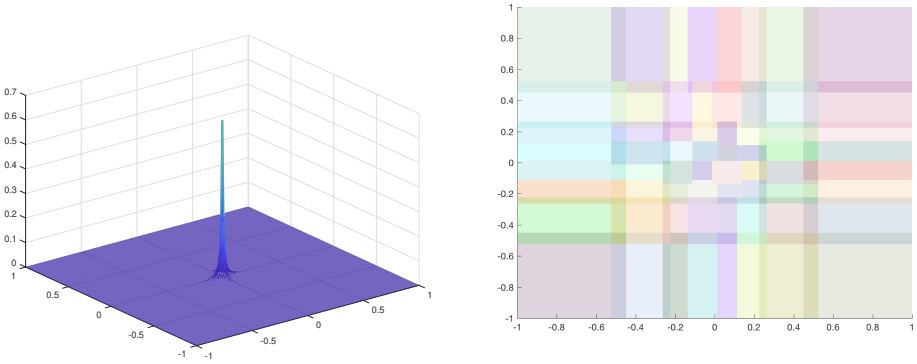
Function	Alg.	Error	Build time	Eval time	Points / Rank
$\log(1+\frac{x_1^2+x_2^4}{10^{-5}})$	T	1.16×10^{-15}	0.525	0.1235	69800
	C	1.14×10^{-6}	2.30	0.10	30
$\arctan(\frac{x_1+x_2^2}{10^{-2}})$	T	1.83×10^{-14}	2.241	0.3590	917515
	C	7.09×10^{-12}	150	5.0	816
$\frac{10^{-4}}{(10^{-4}+x_1^2)(10^{-4}+x_2^2)}$	T	1.86×10^{-15}	0.606	0.0728	117056
	C	5.44×10^{-15}	0.049	0.0037	1
franke	T	$1.33\times10-15$	0.061	0.0069	9270
	C	1.33×10^{-15}	0.020	0.0024	4
$\cos(u_1\pi+\sum_{i=1}^2 a_ix_i)$	T	23.00×10^{-15}	0.007	0.0012	972
	C	4.47×10^{-14}	0.016	0.0020	2
$\prod_{i=1}^2(a_i^{-2}+(x_i-u_i)^2)^{-1}$	T	2.01×10^{-15}	0.063	0.0099	21232
	C	1.59×10^{-12}	0.020	0.0022	1
$(1+\sum_{i=1}^2 a_ix_i)^{-3}$	T	3.33×10^{-16}	0.006	0.0004	25
	C	2.27×10^{-12}	0.012	0.0021	4
$\exp(-\sum_{i=1}^2 a_i^2(x_i-u_i)^2)$	T	$7.77\times10-16$	0.005	0.0012	1862
	C	4.44×10^{-16}	0.015	0.0022	1
square peg	T	2.22×10^{-15}	0.126	0.0264	111188
	C	1.22×10^{-15}	0.023	0.0012	1
tilted peg	T	2.00×10^{-15}	0.214	0.0375	117544
	C	7.68×10^{-14}	0.265	0.0181	100

Table 3.1: Observed error and wall-clock times for the tree-based (T) and Chebfun2 (C) algorithms with target tolerance 10^{-16} and $N = 129$. Build time is for constructing the approximation object, and eval time for evaluating an approximant on a 200x200 uniform grid (all times in seconds). Also shown: for the tree-based method, the total number of stored sampled function values, and for Chebfun2, the numerically determined rank of the function. Here $u = [0.75, 0.25]$ and $a = [5, 10]$.



(a) $\arctan\left((x+y^2)/0.01\right)$ (b) Overlapping subdomains

Figure 3.3: Overlapping subdomains constructed by the adaptive tree method for a function with a nonlinear "cliff."



(a) $\frac{10^{-4}}{(10^{-4}+x^2)(10^{-4}+y^2)}$ (b) Overlapping subdomains

Figure 3.4: Overlapping subdomains constructed by the adaptive tree method for a function with a sharp spike.

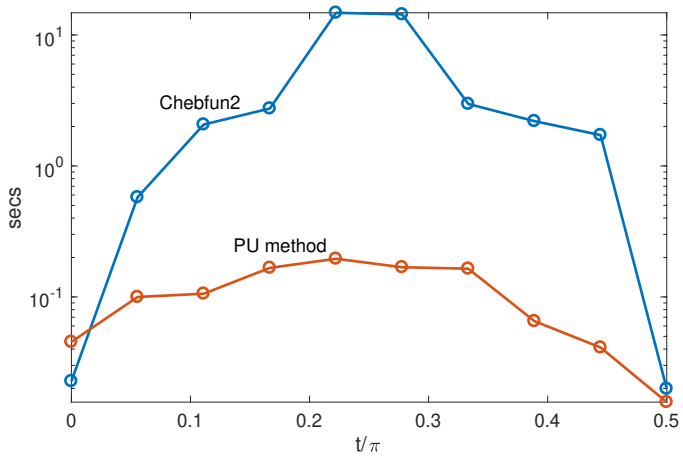


Figure 3.5: Comparison of construction times for $\arctan(250(\cos(t)x + \sin(t)y))$ for $t \in [0, \pi/4]$.

execution time of Chebfun2 varying over nearly three orders of magnitude. While our method is also responsive to the angle of the wave, the variation in time is about half an order of magnitude, and our codes are faster in all but the rank-one case $t = 0$ (for which both methods are fast).

Our next experiment is to add and multiply the rank-one function $\arctan(250x)$ to the plane wave in (3.14). The construction time results are compared for $t \in [0, \pi/2]$ in Figure 3.6. Here the dependence of Chebfun2 on the angle is less severe than in the simple construction, though it is still more pronounced than for our method. More importantly, the absolute numbers for addition in particular with Chebfun2 would probably be considered unacceptable for interactive computation, while our method takes one second at most.

3.4.2 3D experiments

We next test the 3D functions $1/(\cosh(5(x + y + z)))^2$, $\arctan(5(x + y) + z)$, and 3D versions of the smooth functions from the Genz family test package. Table 3.2 shows the construction time, the time taken to evaluate on a $200 \times 200 \times 200$ grid, and the max error on this grid. We observe dramatic construction timing differences in every case: Chebfun3 outperforms the tree-based method for low-Tucker-rank functions, while for the two higher-rank cases, the tree-based method is the clear winner.

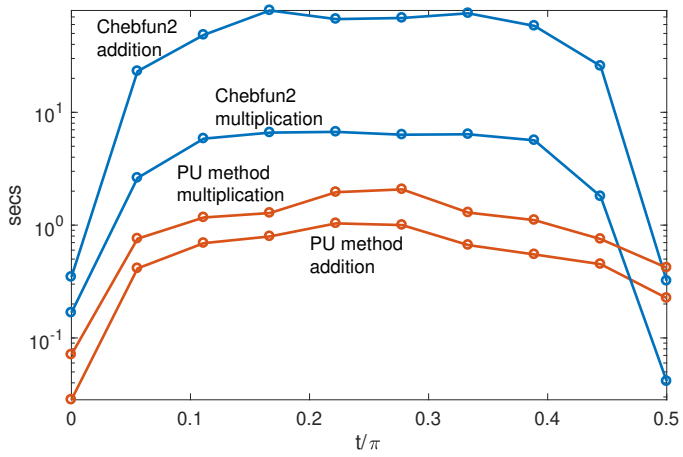


Figure 3.6: Comparison of execution times for multiplication and addition of $\arctan(250x)$ with $\arctan(250(\cos(t)x + \sin(t)y))$ for $t \in [0, \pi/4]$.

Chebfun3 performance is more extreme in both senses, while the tree-based method is more consistent across these examples. Chebfun3 is also faster for evaluation overall, even in high-rank cases, though the evaluation times are typically far less than the construction times.

We repeat our experiment testing the importance of axes alignment using the function

$$\arctan(5(\sin(p)\cos(t)x + \sin(p)\sin(t)y + \cos(p)z)) \tag{3.15}$$

for $p, t \in [0, \pi/4]$. Timing results can be seen in Figure 3.7. As in 2D, the Chebfun low-rank technique shows wide variation depending on the angles, and a large region of long times. The tree-based method is much less sensitive and faster (by as much as two orders of magnitude) except for the purely axes-aligned cases.

3.5 Extension to nonrectangular domains

We now consider approximation over a nonrectangular domain $\Omega \subset \mathbb{R}^d$. In our construction, a leaf node ν whose domain Ω_ν lies entirely within Ω can be treated as before. However, if $\Omega_\nu \cap \Omega \subsetneq \Omega_\nu$, we use a different approximation technique on ν. The refinement criteria of Algorithm 6 are also modified for this situation.

Function	Alg.	Error	Build time	Eval time	Points / Rank
$\cos(u_1\pi + \sum_{i=1}^{3} a_i x_i)$	T	3.16×10^{-14}	2.958	0.240	561495
	C	2.19×10^{-14}	0.460	0.036	2
$\prod_{i=1}^{3}(a_i^{-2} + (x_i - u_i)^2)^{-1}$	T	2.37×10^{-15}	9.917	0.764	7751626
	C	2.63×10^{-15}	0.148	0.030	1
$(1 + \sum_{i=1}^{3} a_i x_i)^{-4}$	T	5.5810^{-16}	0.351	0.020	216
	C	8.93×10^{-16}	0.174	0.021	5
$\exp(-\sum_{i=1}^{2} a_i^2 (x_i - u_i)^2)$	T	1.45×10^{-15}	0.566	0.097	293305
	C	7.80×10^{-16}	0.066	0.018	1
$1/(\cosh(5(x + y + z)))^2$	T	2.00×10^{-15}	4.337	0.325	3450018
	C	3.66×10^{-13}	74.446	0.050	93
$\arctan(5(x + y) + z)$	T	1.95×10^{-15}	0.758	0.145	1132326
	C	3.17×10^{-13}	75.313	0.033	110

Table 3.2: Observed error and wall-clock times for the tree-based (T) and Chebfun3 (C) algorithms with target tolerance 10^{-16} and $N = 65$. Build time is for constructing the approximation object, and eval time for evaluating an approximant on a 200^3 uniform grid (all times in seconds). Also shown: for the tree-based method, the total number of stored sampled function values, and for Chebfun3, the numerically determined rank of the function. Here $u = [0.75, 0.25, -0.75]$ and $a = [25, 25, 25]$.

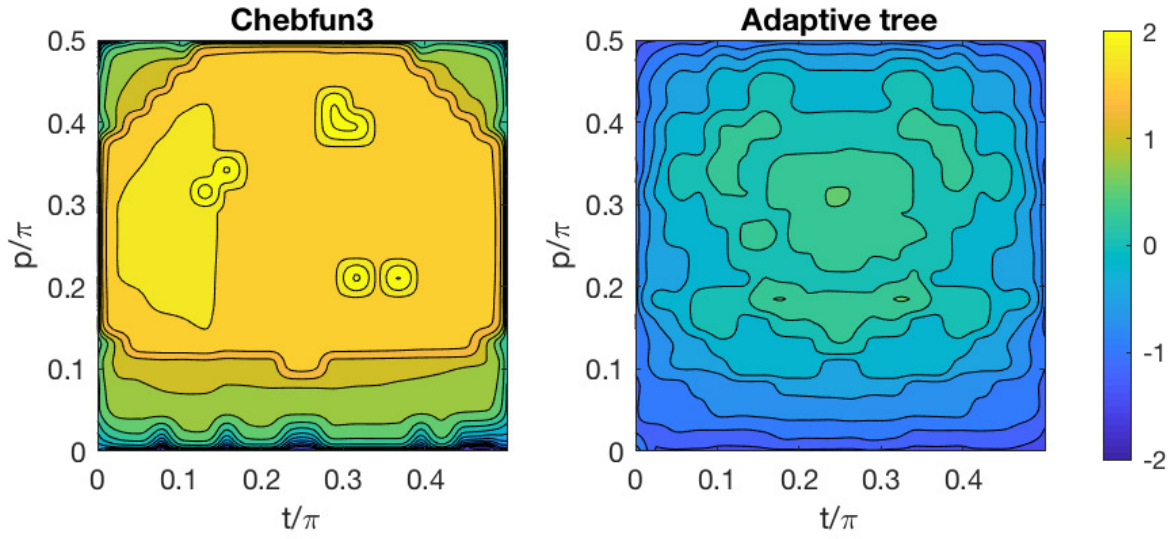


Figure 3.7: Construction time comparison for the 3D function $\arctan(5(\sin(p)\cos(t)x + \sin(p)\sin(t)y + \cos(p)z))$, with varying angles. Colors and contours correspond to the base-10 log of execution time in seconds.

3.5.1 Algorithm modifications

On a leaf whose domain extends outside of Ω, we again use a tensor-product Chebyshev polynomial as in (3.3), but choose its coefficient array C by satisfying a discrete least squares criterion:

$$\arg\min_{C} \sum_{i=1}^{P} \left(f(\mathbf{x}_i) - \tilde{p}(\mathbf{x}_i)\right)^2, \tag{3.16}$$

where $\Xi = \{\mathbf{x}_i\}_{i=1}^{P} \subset \Omega_\nu \cap \Omega$ is a point set in the "active" part of the leaf's domain, $\Omega_\nu \cap \Omega$. In practice we can form a matrix A whose columns are evaluations of each basis function at the points in Ξ, leading to a standard $P \times N^d$ linear least squares problem. We choose Ξ as the part of the standard $(2N)^d$-sized Chebyshev grid lying inside Ω.

This technique resembles Fourier extension or continuation techniques [1, 42], so we refer to it as a *Chebyshev extension approximation*. Unlike the Fourier case, however, there is no real domain extension involved; rather one constrains the usual multivariate polynomial only over part of its usual tensor-product domain. The condition number of A in the Fourier extension case has been shown to increase exponentially with the degree of the approximation [2], because the collection of functions spanning the approximation space is a *frame* rather than a basis. We see the same phenomenon with Chebyshev extension; essentially, constraining the polynomial over only part of the hypercube leaves it underdetermined. To cope with the numerical rank deficiency of A, we rely on the basic least-squares solution computed by the MATLAB backslash. We found this to be as good as or better than the pseudoinverse with a truncated SVD.

We modify Algorithm 7 so that when a domain is split, the resulting zones of the children are shrunk if possible to just contact the boundary of Ω. (An exception is the shared interface between the newly created children, which is fixed.) This helps to keep a substantial proportion of a leaf's domain within Ω.

We also modify how refinement decisions are made and executed in Algorithm 6, for a subtle reason. The original algorithm is able to exploit the very different resolution requirements for a function such as, say, $xT_{60}(y)$, by testing for sufficient resolution in

each dimension independently and splitting accordingly. We find experimentally that if the function is like this over Ω, the extension of it to the unconstrained part of the leaf node's domain has uniform resolution requirements in all variables. Therefore, we use a simpler refinement process: if the norm of the least-squares residual (normalized by $\sqrt{P}$) is not acceptably small, we split in all dimensions successively. In effect, the approximation becomes a quadtree or octree within those nodes that do not lie entirely within Ω.

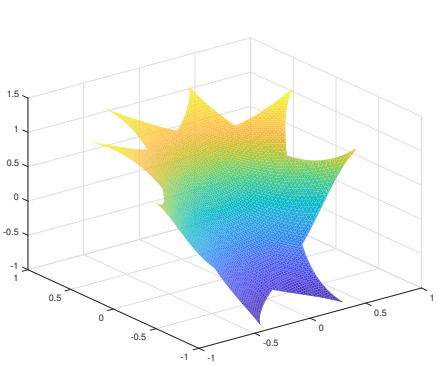
3.5.2 Numerical experiments

We chose the test functions

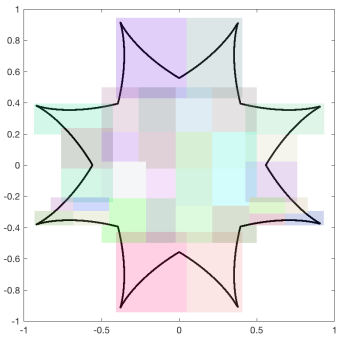
$$\begin{aligned} g_1 &= \exp(x+y), & g_2 &= \frac{1}{((x-1.1)^2)+(y-1.1)^2)^2}, \\ g_3 &= \cos(24x-32y)\sin(21x-28y), & g_4 &= \arctan(3(x^2+y)). \end{aligned} \tag{3.17}$$

We approximated each function on each of three domains: the unit disk, the diamond $|x|+|y| \leq 1$, and the double astroid seen in Figure 3.8. The initial box (root of the approximation tree) was chosen to tightly enclose the given domain. For each test we set $N = 17$ and the target tolerance to 10^{-10}. We timed both the adaptive construction and the evaluation on a 200×200 grid, and recorded the max error as in the previous section. In each case, we choose initial box to fit the domain as tightly as possible. These results can be seen in Table 3.3. The resulting approximation of g_4 on the double astroid is shown in Figure 3.8, along with the adaptively found subdomains.

When the function is smooth or contains localized features, we find that the method is both efficient and highly accurate; in the smoothest case of g_1, a global multivariate least-squares polynomial is sufficient. Only for g_3, which requires uniformly fine resolution throughout the domains, is there a construction time longer than a few seconds. The Fourier extension methods described in [52] are implemented in Julia, making a direct quantitative comparisons difficult, but based on the orders of magnitude of the results reported there, we feel confident that our results for these examples are superior.



(a) Plot of $\arctan(3(y^2+x))$.



(b) Plot of subdomains.

Figure 3.8: Plot of $\arctan(3(y^2+x))$ and the subdomains formed from the partition of unity method. The error in this approximation was found to be about 10^{-11}.

function	domain	error	construct time	interp time	points
g_1	disk	5.44E-15	1.369	0.012	289
	diamond	2.06E-11	0.040	0.002	289
	astroid	2.01E-08	0.071	0.001	289
g_2	disk	2.40E-10	2.558	0.117	3757
	diamond	2.40E-11	0.406	0.012	2023
	astroid	2.14E-10	1.511	0.023	4624
g_3	disk	4.44E-11	11.305	1.500	245650
	diamond	2.35E-11	10.894	0.854	178020
	astroid	1.67E-10	28.072	0.836	153780
g_4	disk	7.49E-11	1.866	0.059	12138
	diamond	1.45E-11	1.536	0.053	9826
	astroid	1.09E-11	3.221	0.049	9826

Table 3.3: Observed error and wall-clock times for the adaptive tree method to approximate the functions given in (3.17) on three different 2D domains. Also shown is the total number of sampled function values stored over all the leaves of each final tree.

3.6 Discussion

For functions over hyperrectangles of uncorrelated variables or that otherwise are well-aligned with coordinate axes, low-rank and sparse-grid approximations can be expected to be highly performant. We have demonstrated an alternative adaptive approach that, in two or three dimensions, typically performs very well on such functions but is far less dependent on that property. Our method sacrifices the use of a single global representation that could achieve true spectral convergence, but in practice we are able to use a partition of unity to construct a smooth, global approximation of very high accuracy in a wide range of examples.

The adaptive domain decomposition offers some other potential advantages we have not yet exploited, but are studying. It offers a built-in parallelism for function construction and evaluation. It allows efficient updating of function values locally, rather than globally, over the domain.

By replacing tensor-product interpolation on the leaves with a simple least-squares approximation using the same multivariate polynomials, we have been able to demonstrate at least reasonable performance in approximation over nonrectangular domains. Further investigation is required to better understand the least-squares approximation process, optimize adaptive strategies, and find efficient algorithms for merging trees and operations such as integration.

Chapter 4

PRECONDITIONED NONLINEAR ITERATIONS FOR OVERLAPPING CHEBYSHEV DISCRETIZATIONS WITH INDEPENDENT GRIDS

4.1 Introduction

For nonlinear problems, the obvious extension of additive Schwarz (AS) preconditioning is to apply it as described above on the linearized equations that are determined by a quasi-Newton iteration. We refer to this process as a *Newton–Krylov–Schwarz* (NKS) procedure, reflecting the nesting order of the different elements of linearization, linear solver, and preconditioning. Cai and Keyes [15] proposed instead modifying the *nonlinear* problem using the Schwarz ansatz. In addition to yielding a preconditioned linearization for the Krylov solver, the preconditioned nonlinear problem exhibited more robust convergence for the Newton iteration than did the original nonlinear problem. They called their method ASPIN, short for *additive Schwarz preconditioned inexact Newton*. As a technical matter, they did not recommend applying the true Jacobian of the system, preferring an approximation that required less effort. Subsequently, Dolean et al. [21] pointed out that Cai and Keyes did not use the RAS form of AS preconditioning, and they proposed an improved variant called RASPEN that does. We refer to this type of nonlinear preconditioning as *Schwarz–Newton–Krylov* (SNK), because the Schwarz ansatz is applied before the linearization begins.

Our interest is in applying the nonlinear preconditioning technique to spectral collocation discretizations in overlapping rectangles or cuboids, leading to globally smooth approximations constructed from a partition of unity [4]. In this context, there is not naturally a single global discretization whose degrees of freedom are partitioned

into overlapping sets, because the Chebyshev (or Legendre, or other classical) nodes will not generally coincide within the overlapping regions. In principle one could link the degrees of freedom within overlap regions by interpolating between subdomains, but this process adds complication, computational time, and (in the parallel context) communication of data.

Here we present an alternative strategy that begins by replacing the original PDE problem with the Schwarz problems on the union of the subdomains. That is, rather than regarding the subdomains as solving the global PDE on a region that includes portions shared with other subdomains, each subdomain has a "private copy" of its entire region and is free to have its own solution values throughout. Of course, the new global problem is not solved until the interface values of every subdomain agree with values interpolated from other subdomains that contain the interface. As a Schwarz starting point, our technique has both NKS and SNK variants.

One advantage of this new formulation is that interpolations need to be done only on lower-dimensional interfaces, rather than throughout the overlap regions. Another is that plain AS is preferred to RAS, because each subdomain has to update its own values separately. We show that it is straightforward to implement exact Jacobians for SNK with nothing more than the ability to do fully local PDE nonlinear and linearized solves, plus the ability to transfer values between subdomains through interface interpolations. We also derive a two-level method to prevent convergence degradation as the number of subdomains increases. The performance of the NKS and SNK methods is validated and compared through several numerical experiments.

4.2 PDE problem and multidomain formulation

The main goal of this work is to solve the PDE

$$\phi(\boldsymbol{x}, u) = 0, \qquad \boldsymbol{x} \in \Omega, \tag{4.1a}$$

$$\beta(\boldsymbol{x}, u) = 0, \qquad \boldsymbol{x} \in \partial\Omega, \tag{4.1b}$$

where $u(\boldsymbol{x})$ is the unknown solution and ϕ and β are nonlinear differential operators (with ϕ being of higher order). (We can easily extend to the case where u, ϕ, and β are vector-valued, but we use scalars to calm the notation.) Many Schwarz-based algorithms for (4.1) begin with a global discretization whose solution is accelerated by an overlapping domain decomposition. In this situation, some of the numerical degrees of freedom are shared by multiple subdomains—either directly or through interpolation—and proper use of additive Schwarz (AS) calls for the restricted-AS (RAS) implementation, which essentially insures that updates of shared values are done only once from the global perspective, not independently by the subdomains.

We take a different approach, replacing the original problem (4.1) with

$$\phi(\boldsymbol{x}, u_i) = 0, \qquad \boldsymbol{x} \in \Omega_i, \qquad i = 1, \ldots, N, \tag{4.2a}$$

$$\beta(\boldsymbol{x}, u_i) = 0, \qquad \boldsymbol{x} \in \Gamma_{i0} = \partial\Omega \cap \partial\Omega_i, \qquad i = 1, \ldots, N, \tag{4.2b}$$

$$u_i = u_j, \qquad \boldsymbol{x} \in \Gamma_{ij} = \partial\Omega_i \cap Z_j, \qquad i, j = 1, \ldots, N, \tag{4.2c}$$

where now $u_1, \ldots, u_N$ are unknown functions on overlapping subdomains Ω_i that cover Ω, and the Z_i are nonoverlapping zones lying within the respective subdomains. Clearly any strong solution of (4.1) is also a solution of (4.2), and while the converse is not necessarily so in principle, we regard the possibility of finding a solution of (4.2) that is not also a solution of (4.1) as remote in practice.

The key consequence of starting from (4.2) as the global problem is that each overlapping region is covered separately by the involved subdomains; each is free to update its representation independently in order to converge to a solution. From one point of view, our discretizations of the overlap regions are redundant and somewhat wasteful. However, the fraction of redundant discrete unknowns is very modest. In return, we only need to interpolate on the interfaces, there is no need to use the RAS formulation, and the coarsening needed for a two-level variant is trivial (see section 4.2.3).

4.2.1 Discretization

We now describe a collocation discretization of (4.2) for concreteness. Each subfunction $u_i(\boldsymbol{x})$ is discretized by a vector $\boldsymbol{u}_i$ of length n_i. By $\boldsymbol{u} = [\![\boldsymbol{u}_i]\!]$ we mean a concatenation of all the discrete unknowns over subdomains $i = 1, \ldots, N$ into a single vector. Subdomain Ω_i is discretized by a node set $X_i \subset \overline{\Omega}_i$ and a boundary node set $B_i \subset \partial\Omega_i$. The total cardinality of X_i and B_i together is also n_i. The boundary nodes are subdivided into nonintersecting sets $G_{ij} = B_i \cap Z_j$ for $j \neq i$, and $G_{i0} = B_i \cap \partial\Omega$.

For each i, the vector $\boldsymbol{u}_i$ defines a function $\tilde{u}_i(\boldsymbol{x})$ on Ω_i. These can be used to evaluate ϕ and β from (4.2) anywhere in Ω_i. We define an n_i-dimensional vector function $\boldsymbol{f}_i$ as the concatenation of three vectors:

$$\boldsymbol{f}_i(\boldsymbol{u}_i) = \begin{cases} \phi(\boldsymbol{x}, \tilde{u}_i) & \text{for all } \boldsymbol{x} \in X_i, \\ \beta(\boldsymbol{x}, \tilde{u}_i) & \text{for all } \boldsymbol{x} \in G_{i0}, \\ \tilde{u}_i(\boldsymbol{x}) & \text{for all } \boldsymbol{x} \in G_{ij},\ j = 1, \ldots, i-1, i+1, \ldots, N. \end{cases} \tag{4.3}$$

In addition, we have the linear *transfer operator* $\boldsymbol{T}_i$ defined by

$$\boldsymbol{T}_i\boldsymbol{u} = \begin{cases} 0 & \text{for all } \boldsymbol{x} \in X_i, \\ 0 & \text{for all } \boldsymbol{x} \in G_{i0}, \\ \tilde{u}_j(\boldsymbol{x}) & \text{for all } \boldsymbol{x} \in G_{ij},\ j = 1, \ldots, i-1, i+1, \ldots, N. \end{cases} \tag{4.4}$$

Note that while $\boldsymbol{f}_i$ is purely local to subdomain i, the transfer operator $\boldsymbol{T}_i$ operates on the complete discretization $\boldsymbol{u}$, as it interpolates from "foreign" subdomains onto the parts of B_i lying inside Ω. Finally, we are able to express the complete discretization of (4.2) through concatenations over the subdomains. Let $\boldsymbol{u} = [\![\boldsymbol{u}_i]\!]$, $\boldsymbol{f}(\boldsymbol{u}) = [\![\boldsymbol{f}_i(\boldsymbol{u}_i)]\!]$, and $\boldsymbol{T}\boldsymbol{u} = [\![\boldsymbol{T}_i\boldsymbol{u}]\!]$. Then the discrete form of (4.2) is the nonlinear equation

$$\boldsymbol{f}(\boldsymbol{u}) - \boldsymbol{T}\boldsymbol{u} = \boldsymbol{0}. \tag{4.5}$$

For a square discretization, the goal is to solve (4.5), while in the least-squares case, the goal is to minimize $\boldsymbol{f}(\boldsymbol{u}) - \boldsymbol{T}\boldsymbol{u}$ in the (possibly weighted) 2-norm.

4.2.2 Newton–Krylov–Schwarz

The standard approach to (4.5) for a large discretization is to apply an inexact Newton iteration with a Krylov subspace solver for finding correcting steps from the linearization. Within the Krylov solver we have a natural setting for applying an AS preconditioner. Specifically, if we have a proposed approximate solution $\boldsymbol{u}$, we evaluate the nonlinear residual $\boldsymbol{r} = \boldsymbol{f}(\boldsymbol{u}) - \boldsymbol{T}\boldsymbol{u}$. We then (inexactly, perhaps) solve the linearization $\big[\boldsymbol{f}'(\boldsymbol{u}) - \boldsymbol{T}\boldsymbol{u}\big]\boldsymbol{s} = -\boldsymbol{r}$ for the Newton correction $\boldsymbol{s}$, using a Krylov solver such as GMRES. These iterations are preconditioned by the block diagonal matrix $\boldsymbol{f}'(\boldsymbol{u})$, which is simply the block diagonal of the subdomain Jacobians $\boldsymbol{f}_i'(\boldsymbol{u}_i)$. We refer to this method as *Newton–Krylov–Schwarz*, or NKS.

Implementation of NKS requires three major elements: the evaluations of $\boldsymbol{f}(\boldsymbol{u})$ and $\boldsymbol{T}\boldsymbol{u}$ for given $\boldsymbol{u}$, the application of the Jacobian $\boldsymbol{f}'(\boldsymbol{u})$ to a given vector $\boldsymbol{v}$, and the inversion of $\boldsymbol{f}'(\boldsymbol{u})$ for given data. All of the processes involving $\boldsymbol{f}$ are embarrassingly parallel and correspond to standard steps in solving the PDE on the local subdomains. Each application of the transfer operator $\boldsymbol{T}$, however, requires a communication from each subdomain to its overlapping neighbors, as outlined in Algorithm 9. This step occurs once in evaluating the nonlinear residual and in every GMRES iteration to apply the Jacobian. In a parallel code, the communication steps could be expected to be a major factor in the performance of the method.

Algorithm 9 Apply transfer operator, $\boldsymbol{T}\boldsymbol{u}$.

```
Interpret input u as concatenated [[u_i]].
for j = 1, ..., N (in parallel) do
  for all neighboring subdomains i do
    Evaluate ũ_j at x ∈ G_ij.
  end for
end for
```

4.2.3 Two-level scheme

As is well known [22], AS schemes should incorporate a coarse solution step in order to maintain convergence rates as the number of subdomains increases. The

Algorithm 10 Evaluate FAS correction $\boldsymbol{c}(\boldsymbol{u})$.

Apply Algorithm 9 to compute $\boldsymbol{T}\boldsymbol{u}$.
Compute (in parallel) $\hat{\boldsymbol{u}} = \boldsymbol{R}\boldsymbol{u}$.
Compute (in parallel) $\hat{\boldsymbol{r}} = \boldsymbol{R}(\boldsymbol{f}(\boldsymbol{u}) - \boldsymbol{T}\boldsymbol{u}) - \hat{\boldsymbol{f}}(\hat{\boldsymbol{u}})$.
Solve equation (4.6) for $\hat{\boldsymbol{e}}$.
Compute (in parallel) the prolongation $\boldsymbol{P}\hat{\boldsymbol{e}}$.

methods described above depend on the subdomain discretization sizes n_i of the collocation nodes and solution representation, respectively. Now suppose we decrease the discretization sizes to $\hat{n}_i$, and denote the corresponding discretizations of (4.5) by $\hat{\boldsymbol{f}}(\hat{\boldsymbol{u}}) - \hat{\boldsymbol{T}}\hat{\boldsymbol{u}} = \boldsymbol{0}$. We can define a restriction operator $\boldsymbol{R}$ that maps fine-scale vectors to their coarse counterparts. This operator is block diagonal, i.e., it can be applied independently within the subdomains. We can also construct a block diagonal prolongation operator $\boldsymbol{P}$ for mapping the solution representation from coarse to fine scales.

We are then able to apply the standard Full Approximation Scheme (FAS) using the coarsened problem [12]. Specifically, we solve the coarse problem

$$\hat{\boldsymbol{f}}(\hat{\boldsymbol{e}} + \boldsymbol{R}\boldsymbol{u}) - \hat{\boldsymbol{T}}\hat{\boldsymbol{e}} - \hat{\boldsymbol{f}}(\boldsymbol{R}\boldsymbol{u}) + \boldsymbol{R}(\boldsymbol{f}(\boldsymbol{u}) - \boldsymbol{T}\boldsymbol{u}) = \boldsymbol{0} \tag{4.6}$$

for the coarse correction $\hat{\boldsymbol{e}}$, and define $\boldsymbol{c}(\boldsymbol{u}) = \boldsymbol{P}\hat{\boldsymbol{e}}$ as the FAS corrector at the fine level. The procedure for calculating $\boldsymbol{c}$ is outlined in Algorithm 10.

We also require the action of the Jacobian $\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{u}} = \boldsymbol{P}\frac{\partial \hat{\boldsymbol{e}}}{\partial \boldsymbol{u}}$ on a given vector $\boldsymbol{v}$. It is straightforward to derive from (4.6) that

$$\big[\hat{\boldsymbol{f}}'(\hat{\boldsymbol{e}} + \boldsymbol{R}\boldsymbol{u}) - \hat{\boldsymbol{T}}\big]\frac{\partial \hat{\boldsymbol{e}}}{\partial \boldsymbol{u}} = -\big(\hat{\boldsymbol{f}}'(\hat{\boldsymbol{e}} + \boldsymbol{R}\boldsymbol{u}) - \hat{\boldsymbol{f}}'(\boldsymbol{R}\boldsymbol{u})\big)\boldsymbol{R} - \boldsymbol{R}\big(\boldsymbol{f}'(\boldsymbol{u}) - \boldsymbol{T}\big). \tag{4.7}$$

Note that the matrix $\hat{\boldsymbol{f}}'(\hat{\boldsymbol{e}} + \boldsymbol{R}\boldsymbol{u})$ should be available at no extra cost from the end of the Newton solution of (4.6). Algorithm 11 describes the corresponding algorithm for computing the application of $\boldsymbol{c}'(\boldsymbol{u})$ to any vector $\boldsymbol{v}$. Even though $\boldsymbol{c}'$ is of the size of the fine discretization, the computation requires only coarse-dimension dense linear algebra.

Algorithm 11 Apply Jacobian $\boldsymbol{c}'(\boldsymbol{u})$ for the FAS corrector to a vector $\boldsymbol{v}$.

Apply Algorithm 10 to compute $\hat{\boldsymbol{e}}$, $\hat{\boldsymbol{u}}$, and the final value of $\hat{\boldsymbol{A}} = \hat{\boldsymbol{f}}'(\hat{\boldsymbol{e}} + \hat{\boldsymbol{u}})$.
Apply Algorithm 9 to compute $\boldsymbol{T}\boldsymbol{v}$.
Set (in parallel) $\hat{\boldsymbol{r}} = \boldsymbol{R}\big(\boldsymbol{f}'(\boldsymbol{u})\boldsymbol{v} - \boldsymbol{T}\boldsymbol{v}\big)$ and $\hat{\boldsymbol{v}} = \boldsymbol{R}\boldsymbol{v}$.
Set (in parallel) $\hat{\boldsymbol{b}} = \hat{\boldsymbol{A}}\hat{\boldsymbol{v}} - \hat{\boldsymbol{f}}'(\hat{\boldsymbol{u}})\hat{\boldsymbol{v}} + \hat{\boldsymbol{r}}$.
Solve the linear system $(\hat{\boldsymbol{A}} - \hat{\boldsymbol{T}})\hat{\boldsymbol{y}} = -\hat{\boldsymbol{b}}$ for $\hat{\boldsymbol{y}}$.
Compute (in parallel) $\boldsymbol{P}\hat{\boldsymbol{y}}$.

Finally, we describe how to combine coarsening with the preconditioned fine scale into a two-level algorithm. If we were to alternate coarse and fine corrections in the classical fixed-point form,

$$\begin{aligned}\boldsymbol{u}^{\dagger} &= \boldsymbol{u} + \boldsymbol{c}(\boldsymbol{u}),\\ \boldsymbol{u}^{\text{new}} &= \boldsymbol{u}^{\dagger} + \boldsymbol{f}(\boldsymbol{u}^{\dagger}) - \boldsymbol{T}\boldsymbol{u}^{\dagger},\end{aligned}$$

then we are effectively seeking a root of

$$\boldsymbol{h}(\boldsymbol{u}) := \boldsymbol{c}(\boldsymbol{u}) + (\boldsymbol{f} - \boldsymbol{T})(\boldsymbol{u} + \boldsymbol{c}(\boldsymbol{u})). \tag{4.8}$$

Finally, the Jacobian of the combined map is straightforwardly

$$\boldsymbol{h}'(\boldsymbol{u}) = \boldsymbol{c}'(\boldsymbol{u}) + (\boldsymbol{f}' - \boldsymbol{T})(\boldsymbol{u} + \boldsymbol{c}(\boldsymbol{u})) \cdot (\boldsymbol{I} + \boldsymbol{c}'(\boldsymbol{u})). \tag{4.9}$$

Thus the action of $\boldsymbol{h}'$ on a vector can be calculated using the algorithms for $\boldsymbol{c}'$, $\boldsymbol{f}'$, and $\boldsymbol{T}$.

4.3 Preconditioned nonlinear iterations

As shown in section 4.2.2, the inner Krylov iterations of the NKS method are governed by the preconditioned Jacobian $\boldsymbol{I} - [\boldsymbol{f}'(\boldsymbol{v})]^{-1}\boldsymbol{T}$. Following the observation of Cai and Keyes [15], we next derive a method that applies Krylov iterations to the same matrix, but arising as the natural result of preconditioning the *nonlinear* problem. Specifically, we precondition (4.5) by finding a root of the nonlinear operator

$$\boldsymbol{g}(\boldsymbol{u}) := \boldsymbol{u} - \boldsymbol{f}^{-1}(\boldsymbol{T}\boldsymbol{u}). \tag{4.10}$$

Evaluation of $\boldsymbol{g}$ is feasible because of the block diagonal (that is, fully subdomain-local) action of the nonlinear $\boldsymbol{f}$. Since we are therefore applying the Schwarz ansatz even before linearizing the problem, we refer to the resulting method as *Schwarz–Newton–Krylov* (SNK).

We have several motivations for a method based on (4.10). First, one hopes that the nonlinear problem, being a (low-rank) perturbation of the identity operator, is somehow easier to solve by Newton stepping than the original form is. Second, the inversion of $\boldsymbol{f}$ means solving independent nonlinear problems on the subdomains with no communication, which well exploits parallelism. Finally, the same structure means that problems with relatively small highly active regions could be isolate the need to solve a nonlinear problem to that region, rather than having it be part of a fully coupled global nonlinear problem.

An algorithm for evaluating $\boldsymbol{g}$ is given in Algorithm 12. It requires one communication between subdomains to transfer interface data, followed by solving (in parallel if desired) the nonlinear subdomain problems $\boldsymbol{f}_i$ defined in (4.3). Note that the local problem in Ω_i is a discretization of the PDE with zero boundary data on the true boundary Γ_{i0} and values transferred from the foreign subdomains on the interfaces.

Algorithm 12 Evaluate SNK residual $\boldsymbol{g}(\boldsymbol{u})$.

Apply Algorithm 9 to compute $\boldsymbol{T}\boldsymbol{u}$.
for $i = 1, \dots, N$ (in parallel) **do**
  Solve $\boldsymbol{f}_i(\boldsymbol{u}_i - \boldsymbol{z}_i) = \boldsymbol{T}_i\boldsymbol{u}$ for $\boldsymbol{z}_i$.
end for
Return $[\![\boldsymbol{z}_i]\!]$.

Using the notation of Algorithm 12, we have that

$$\boldsymbol{f}_i'(\boldsymbol{u}_i - \boldsymbol{z}_i)\left[I - \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{u}}\right] = \boldsymbol{T}_i,$$

which implies that applying $[\partial \boldsymbol{z}_i / \partial \boldsymbol{u}]$ to a vector requires a single linear solve on a subdomain, with a matrix that is presumably already available at the end of the local Newton iteration used to compute $\boldsymbol{g}$. The process for applying $\boldsymbol{g}'(\boldsymbol{u})$ to a vector is outlined in Algorithm 13.

Algorithm 13 Apply Jacobian $\boldsymbol{g}'(\boldsymbol{u})$ to vector $\boldsymbol{v}$ for the SNK problem.

Apply Algorithm 9 to compute $\boldsymbol{T}\boldsymbol{v}$.
for $i = 1, \ldots, N$ (in parallel) **do**
  Solve $[\boldsymbol{f}_i'(\boldsymbol{u}_i - \boldsymbol{z}_i)]\boldsymbol{y}_i = \boldsymbol{T}_i\boldsymbol{v}$ for $\boldsymbol{y}_i$.
end for
Return $[\![\boldsymbol{v}_i - \boldsymbol{y}_i]\!]$.

4.3.1 Two-level scheme

The SNK method can be expected to require coarse correction steps to cope with a growing number of subdomains. An obvious approach to incorporating a coarse-grid correction step is to apply FAS directly, i.e., using the analog of (4.6) with fine $\boldsymbol{g}$ and coarse $\hat{\boldsymbol{g}}$. However, doing so means inverting $\hat{\boldsymbol{g}}$, which introduces another layer of iteration in the overall process.

We have found that it is simpler and successful to apply the FAS correction in the form of the original NKS method, as given in section 4.2.3. All we need to do is replace $\boldsymbol{f}(\boldsymbol{u}) - \boldsymbol{T}\boldsymbol{u}$ and $\boldsymbol{f}' - \boldsymbol{T}$ in (4.8) and (4.9) by $\boldsymbol{g}$ and $\boldsymbol{g}'$, respectively.

4.4 Integration with ode15s

The `ode15s` function in MATLAB that finds the solution to initial value problems

$$M(t, \boldsymbol{x})u' = \phi(t, \boldsymbol{x}, u), \qquad \boldsymbol{x} \in \Omega \tag{4.11}$$

on a time interval $[t_0, t_f]$, given initial values $u(t_0) = u_0$ and mass matrix $M(t, \boldsymbol{x})$. For a step from (t_n, u_n) to (t_{n+1}, u_{n+1}), ode15s finds u_{n+1} that solves

$$M(t_{n+1}, \boldsymbol{x}) \sum_{m=1}^{k} \frac{1}{m} \nabla^m u_{n+1} - h\phi(t_{n+1}, \boldsymbol{x}, u_{n+1}) = 0 \qquad \boldsymbol{x} \in \Omega \tag{4.12}$$

with Newton's method [61], where the initial solution is

$$u_{n+1}^{(0)} = \sum_{m=0}^{k} \nabla^m u_n. \tag{4.13}$$

With $\gamma_k = \sum_{j=1}^{k} \frac{1}{j}$ we have the identity

$$\sum_{m=1}^{k} \frac{1}{m} \nabla^m u_{n+1} = \gamma_k (u_{n+1} - u_n^{(0)}) + \sum_{m=1}^{k} \frac{1}{m} \gamma_m \nabla^m u_n. \tag{4.14}$$

If we let

$$\Psi = \frac{1}{\gamma_k} \left(\sum_{m=1}^{k} \frac{1}{m} \gamma_m \nabla^m u_n - u_n^{(0)} \right) \tag{4.15}$$

then (4.12) can be written as

$$M(t_{n+1}, \boldsymbol{x}) u_{n+1} - \frac{h}{\gamma_k} \phi(t_{n+1}, \boldsymbol{x}, u_{n+1}) + M(t_{n+1}, \boldsymbol{x}) \Psi = 0 \qquad \boldsymbol{x} \in \Omega. \tag{4.16}$$

4.4.1 Discretization

We adapted the `ode15s` function to solve (4.16) with the NKS and SNK methods by discretizing (4.16) at the local subdomains in a similar manner to (4.3). First we define a vector function $\hat{\boldsymbol{f}}_i$ as the following concatenation:

$$\hat{\boldsymbol{f}}_i(\boldsymbol{u}_i, t) = \begin{cases} -\frac{h}{\gamma_k} \phi(t_{n+1}, \boldsymbol{x}, \tilde{u}_i) & \text{for all } \boldsymbol{x} \in X_i \cup G_{i0}, \\ \tilde{u}_i(\boldsymbol{x}) & \text{for all } \boldsymbol{x} \in G_{ij},\ j = 1, \ldots, i-1, i+1, \ldots, N. \end{cases} \tag{4.17}$$

Suppose $\hat{M}_i(t, \boldsymbol{x})$ is the mass matrix from (4.11) defined for the local subdomain of Ω_i. We the define mass matrices $M_i(t, \boldsymbol{x})$ such that

$$M_i(t, \boldsymbol{x})_{mn} = \begin{cases} \hat{M}_i(t, \boldsymbol{x})_{mn}, & \text{for all } \boldsymbol{x} \in X_i \cup G_{i0} \\ 0 & \text{for all } \boldsymbol{x} \in G_{ij},\ j = 1, \ldots, i-1, i+1, \ldots, N \end{cases} \tag{4.18}$$

which treats the interface conditions of (4.2) as algebraic within the initial value problem. We finally define vector functions $\boldsymbol{f}_i$ such that

$$\boldsymbol{f}_i(\boldsymbol{u}_i, t) = M_i(t, \boldsymbol{x}) \boldsymbol{u}_i + M_i(t, \boldsymbol{x}) \tilde{\Psi}_i + \hat{\boldsymbol{f}}_i(\boldsymbol{u}_i, t). \tag{4.19}$$

Defining $\boldsymbol{f}(\boldsymbol{u}, t) = [\![\boldsymbol{f}_i(\boldsymbol{u}_i, t)]\!]$, we have that finding $\boldsymbol{u}$ such that

$$\boldsymbol{f}(\boldsymbol{u}, t) - T(\boldsymbol{u}) = 0 \tag{4.20}$$

will effectively solve for (4.16). We utilize this technique to solve for a time dependent model for the tearfilm thickness of the eye in section 5.3.

4.5 Numerical experiments

For all experiments we compared three methods: NKS, SNK, and the two-level SNK2. The local nonlinear problems for SNK, and the coarse global problems in SNK2, were solved using `fsolve` from the Optimization Toolbox. Each solver used an inexact Newton method as the outer iteration, continued until the residual was less than 10^{-10} relative to the initial residual. For the inner iterations we used MATLAB's `gmres` to solve for the Newton step s_k such that

$$\|F(x_k) + F'(x_k)s_k\| \leq \eta_k \|F(x_k)\| \tag{4.21}$$

where $\eta_0 = 10^{-4}$ and

$$\eta_k = 10^{-4} \left(\frac{\|F(x_k)\|}{\|F(x_{k-1})\|} \right)^2. \tag{4.22}$$

Given certain conditions on $F(x)$, if the intial solution is close enough to the true solution then this set of tolerances will yield a sequence with near q-2 convergence [28].

4.5.1 Regularized driven cavity flow

The first example is a regularized form of the lid-driven cavity flow problem [40], where we replace the boundary conditions with infinitely smooth ones. Using the velocity-vorticity formulation, in terms of the velocity u, v and vorticity ω on $\Omega = [0,1]^2$ we have the nondimensionalized equations

$$\begin{aligned} -\Delta u - \frac{\partial \omega}{\partial y} &= 0, \\ -\Delta v + \frac{\partial \omega}{\partial x} &= 0, \\ -\frac{1}{\mathrm{Re}}\Delta \omega + u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} &= 0, \end{aligned} \tag{4.23}$$

where Re is the Reynolds number. In the context of this chapter, the Reynolds number can be thought of as a parameter controlling the non-linearity of (4.23). The lid-driven cavity flow problem is a standard test problem for Navier Stokes code [11]. The standard problem specifies discontinuous boundary conditions ($u(x,y) = 1$ at $y = 1$, $u = 0$ elsewhere), causing singularities in the top two corners. Since we are

using Chebyshev approximations we must use regularized boundary conditions that are continuous. On the boundary $\partial\Omega$ we apply

$$\begin{aligned} u &= \begin{cases} \exp\left(\frac{\left(\frac{y-1}{0.1}\right)^2}{1-\left(\frac{y-1}{0.1}\right)^2}\right), & y > b, \\ 0, & y \leq b, \end{cases} \\ v &= 0, \\ \omega &= -\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}, \end{aligned} \tag{4.24}$$

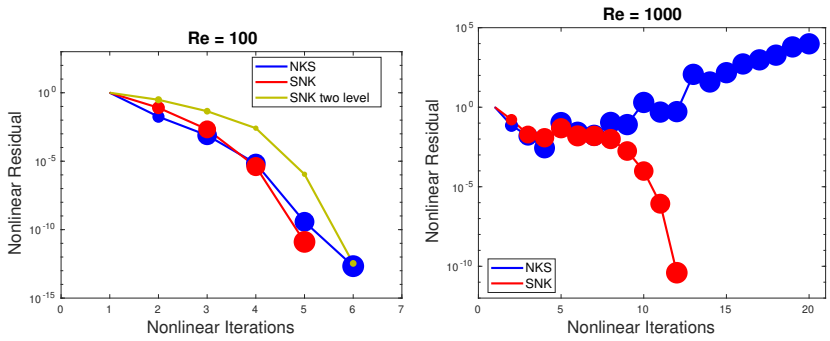
similar to the boundary conditions in [62]. We continuously drop u to 0 with a bump function, with $b = 0.9$. When b is moved closer to 1, a higher resolution is needed near $y = 1$ to compensate for the steeper gradients the bump function produces.

We divided Ω into 16 overlapping patches of equal size (i.e. a 4 by 4 patch structure, as seen in 4.3). Each subdomain was discretized by a second-kind Chebyshev grid of length 33 in each dimension. For the initial guess to the outer solver iterations we extended the boundary conditions (4.24) to all values of x in the square Ω.

The convergence of the three solvers is shown for Re = 100, and SNK and NKS for Re = 1000 in Figure ~~4.1, as well as the plot of the solutions in~~ Figure 4.2. All three methods converge for Re = 100. The number of GMRES iterations per nonlinear iteration are similar for the NKS and SNK methods; this is to be expected since the linear system used to solve the Newton step is similar in both methods. We do however see a dramatic reduction in the number of GMRES iterations with the two-level SNK method.

It is worth noting well that the computational time for each outer iteration varies greatly between solvers. The NKS residual requires only evaluating the discretized PDE and is thus is a good deal faster per iteration than the SNK solvers, which require solving the local nonlinear problems. In addition SNK2 must solve a global coarse problem in each outer iteration, but this added relatively little computing time.

For the higher Reynolds number Re = 1000 we find that while SNK still converges, NKS does not, similar to what was reported in [15]. We also found that the



(a) Nonlinear residuals with $Re = 100$

(b) Nonlinear residuals with $Re = 1000$

Figure 4.1: Nonlinear residuals, normalized by the residual of the initial guess, of the NKS, SNK, and SNK2 solvers on the regularized cavity flow problem (4.23)–(4.24). The area of each marker is proportional to the number of inner GMRES iterations taken to meet the inexact Newton criterion.

(a) Velocity plot with $Re = 100$

(b) Velocity plot with $Re = 1000$

Figure 4.2: Velocity plot of solution to (4.23)–(4.24), with $Re = 100, 1000$.

coarse-level solver in SNK2 had trouble converging.

4.5.2 Burgers equation

The second test problem is Burgers' equation,

$$\nu \Delta u - \nabla \cdot u = 0, \tag{4.25}$$

on $\Omega = [-1, 1]^2$, with Dirichlet boundary condition

$$u = \arctan\left(\cos\left(\frac{3\pi}{16}\right) x + \sin\left(\frac{3\pi}{16}\right) y\right). \tag{4.26}$$

This PDE was solved using a subdomain structure adapted to the function

$$\exp\left(\frac{1}{1 - x^{-20}} + \frac{1}{1 - y^{-20}}\right)$$

Figure 4.3: Subdomains for the cavity flow experiments.

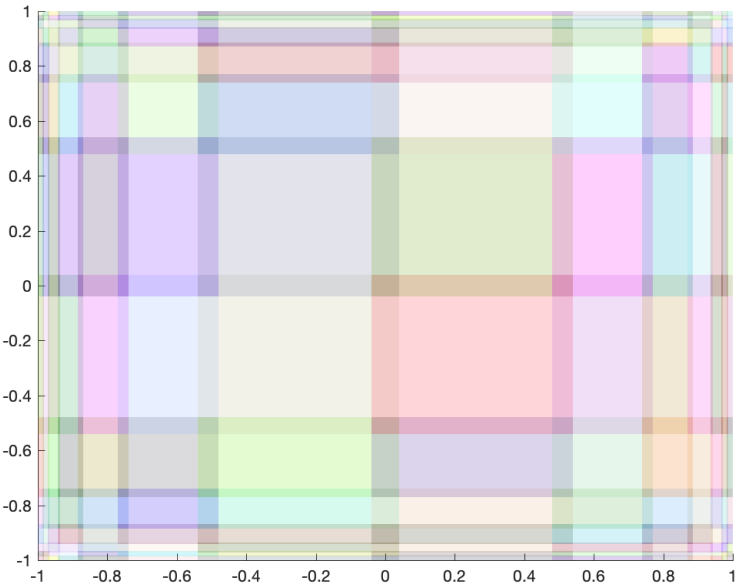


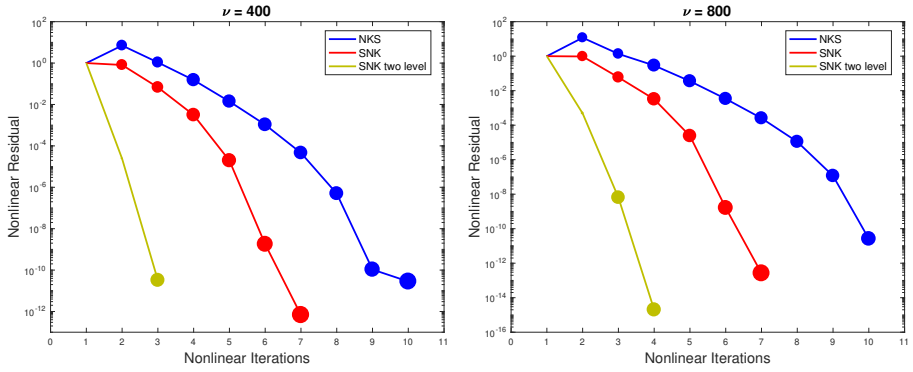
Figure 4.4: Subdomains for the Burgers experiments, found by adapting to the function $\exp(-x^{20}/(1-x^{20}))\exp(-y^{20}/(1-y^{20}))$ in order to increase resolution in the boundary layer.

using the methods in [4], in order to help capture the boundary layers, as shown in Figure 4.4. For the initial guess of the outer iterations, the boundary condition (4.26) was extended throughout Ω.

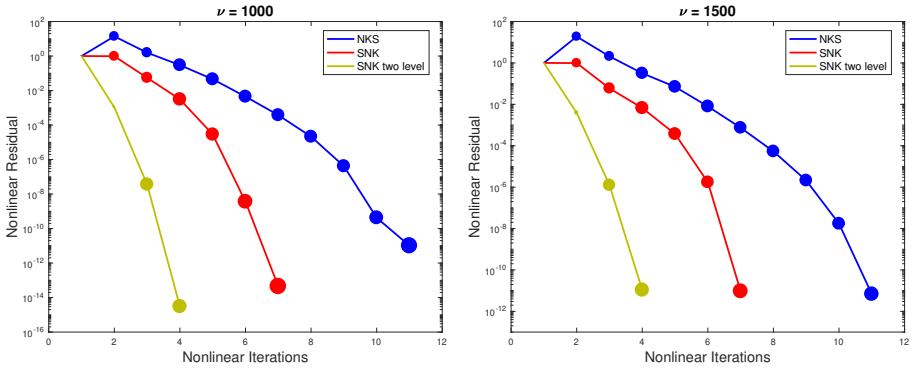
Convergence histories for (4.25)–(4.26) for $1/\nu = 400, 800, 1000, 1500$ are given in Figure 4.5. We observe again that the SNK and SNK2 solvers seem quite insensitive to the diffusion strength, while the number of outer iterations in NKS increases mildly as diffusion wanes. Furthermore, SNK2 converges in about half as many outer iterations as SNK.

4.5.3 Parallel efficiency

A fully parallel-aware implementation of the methods would presumably distribute all the data and solving steps across cores, which would handle communication



(a) Nonlinear residuals with $\nu = 1/400$ (b) Nonlinear residuals with $\nu = 1/800$



(c) Nonlinear residuals with $\nu = 1/1000$(d) Nonlinear residuals with $\nu = 1/1500$

Figure 4.5: Nonlinear residuals, normalized by the residual of the initial guess, of the NKS, SNK, and SNK2 methods to solve (4.25)–(4.26). The area of each marker is proportional to the number of inner GMRES iterations taken to meet the inexact Newton criterion.

Number of cores	Total time (sec.)	Speedup	Jacobian time	Residual time	Residual speedup
1	62.3	—	11.6	42.0	—
2	43.5	1.43	12.7	24.1	1.74
4	33.4	1.86	11.3	15.5	2.71
6	30.0	2.08	11.5	12.1	3.47
8	28.3	2.21	11.3	10.5	4.00
12	27.6	2.26	11.9	9.3	4.51
16	28.6	2.18	13.2	8.8	4.78
20	28.2	2.21	13.1	8.3	5.04

Table 4.1: Parallel timing results for the Burgers equation experiment. "Jacobian time" is the total time spent within applications of the Jacobian to a given vector, and "Residual time" is the total amount of time spent evaluating the nonlinear SNK residual.

of interface values with neighbors when necessary. A simpler step was to modify our serial MATLAB implementation to use the `parfor` capability of the Parallel Computing Toolbox for the most compute-intensive loop in the SNK methods, that for the independent local nonlinear solves in the SNK residual evaluation, followed by factoring the final Jacobian matrices of these solutions. While we also tried parallelizing the loop for applying the inverses of the local Jacobians as part of the full Jacobian application in an inner Krylov iteration, the effect on timings was minimal or even detrimental due to the greater importance of communication relative to that computation.

The parallel SNK2 solver was applied to the Burgers experiment as described in section 4.5.2, but with the domain Ω split into an 8-by-8 array of uniformly sized subdomains, resulting in 3 outer iterations and a total of 18 inner iterations. The code was run on a compute node equipped with two 18C Intel E5-2695 v4 (for 36 total cores), 32 GB of DDR4 memory, and a 100 Gbps Intel OmniPath cluster network. The timing results for different numbers of parallel computing cores are given in Table 4.1. There is a good amount of speedup in evaluations of the nonlinear SNK residuals, consisting mainly of the solution of local nonlinear problems, which dominate the computing time for a small number of cores. However, the parallel efficiency is limited by the other parts of the implementation, most notably the Jacobian evaluations.

4.6 Discussion

We have described a framework for overlapping domain decomposition in which overlap regions are discretized independently by the local subdomains, even in the formulation of the global problem. Communication between subdomains occurs only via interpolation of values to interface points. This formulation makes it straightforward to apply high-order or spectral discretization methods in the subdomains and to adaptively refine them.

The technique may be applied to precondition a linearized PDE, but it may also be used to precondition the nonlinear problem before linearization, to get what we call the Schwarz–Newton–Krylov (SNK) technique. In doing so, one gets the same benefit of faster Krylov inner convergence, but the resulting nonlinear problem is demonstrably easier to solve in terms of outer iterations and robustness. Although we have not given the derivation here, the Jacobian of the preconditioned nonlinear problem is readily shown to be a low-rank perturbation of the identity. Thus Kantorovich or other standard convergence theory for Newton's method [20] may therefore suggest improved local convergence rates and larger basin of attraction. We have not yet pursued this analysis.

We have demonstrated that the SNK method can easily be part of a two-level Full Approximation Scheme in order to keep iteration counts from growing as the number of subdomains grows. The coarse level is simply a coarsening on each subdomain, so that restriction and prolongation steps can be done simply and in parallel. Indeed, the situation should make a fully multilevel implementation straightforward, as the multilevel coarsenings and refinements can all be done within subdomains.

The most time-consuming part of the SNK algorithm is expected to be typically in the solution of nonlinear PDE problems within each subdomain using given boundary data. These compute-intensive tasks require no communication and are therefore efficient to parallelize. By contrast, each inner Krylov iteration (i.e., Jacobian application) of both SNK and linearly preconditioned NKS requires a communication of interface data between overlapping subdomains, which appears to generate a more

communication-bound form of parallelism. An additional feature of the SNK approach, mentioned also in [15], is that subdomains of low solution activity can be expected to be found relatively quickly. We observed this to be the case in the cavity flow problem of section 4.5.1, where local solutions in regions of low activity were sometimes 3-4 times faster as those in regions with steep solution gradients. This presents a natural way to limit the spatial scope of difficult nonlinear problems, though it also raises questions for load balancing in a parallel environment.

Finally, we remark that an important extension in [4] is to use least-squares approximation rather than interpolation to incorporate nonrectangular (sub)domains. We have been able to write a least-squares (as opposed to collocation) generalization of SNK and test it in one dimension, but much further work is required.

Chapter 5

APPLICATION TO BLINKING EYE SIMULATION

5.1 Introduction

The tear film is a thin layer of liquid film on the ocular surface spread through blinking. It serves several vital functions including protecting the ocular surface with moisture, transporting waste, and as providing a smooth ocular surface. Within each blink cycle, a tear film that is functioning properly will maintain a balance between tear secretion and loss. [41].

Dry eye syndrome (DES) is a collection of symptoms that include blurred vision, burning, foreign body sensation, and tearing and inflammation of the ocular surface. According to a study from 2007, an estimated 4.91 million Americans suffer from DES [14]. A malfunctioning or deficient tear film is thought to be a cause of DES [55], causing the ocular tear film comunity to take interest in the function of the tear film [43] as well as the connection between DES and tear film volume, evaporation [14].

The tear film is composed of three layers, as shown in Figure 5.1. The anterior surface in contact with the air is an oily lipid layer which helps reduce evaporation [56, 54]. The middle aqueous layer is composed mostly of water, and serves to lubricate the eye, move particles and prevent infections. On the surface of the cornea sits the Glycocalyx layer, a forest of proteins that moisturize the corneal surface by attracting water [34].

In 2008, Maki used an overset grid method to study the effects of the tear film under reflex tearing [49]. In an overset grid method, a PDE is discretized and solved on a composite of grids to handle complex geometries and localized features of the function, where the overlapping grids are coupled through interpolation. Maki used

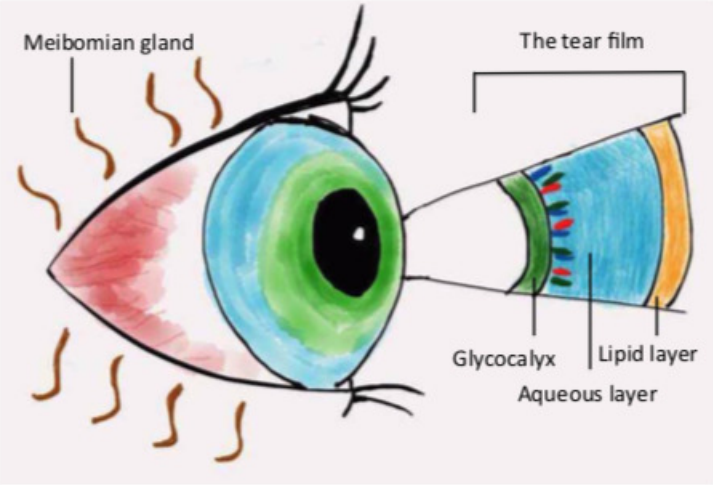


Figure 5.1: Illustration of tear film with labeled layers [74].

fine boundary grids near the upper and lower eyelids in a 1D model to capture localized capillary thinning [49].

Maki extend this technique in 2D allowing for numerical simulations over a realistic open eye domain [48, 50]. Using the Overture framework from Lawrence Livermore National Laboratory [17, 38], she generated a set of grids adapted to the geometry of the eye [17]; an example of this discretization can be seen in Figure 5.2. It was found that the geometry, in conjunction with a tear film thickness boundary condition, produces stronger capillary action in the vicinities of the nasal and temporal canthi [48]. Maki later incorporated flux boundary conditions that depended both on space, including no-flux boundary conditions on a realistic eye-shaped domain as well as non zero flux conditions that incorporated the lacrimal gland influx and puncta drainage [50]. Li built on top these models to include time dependent flux boundary conditions [47], as well as ~~developed~~ new models to study the dynamics of osmolarity within the tear film [46, 45].

In 2007 Heryudono and Braun developed a one-dimensional blinking model of the eye, which included flux conditions of an average supply from the lacrimal gland and drainage due to the puncta [39]. Deng and Driscoll extended this one dimensional model to incorporate a cooling effect, where Chebyshev polynomial approximations

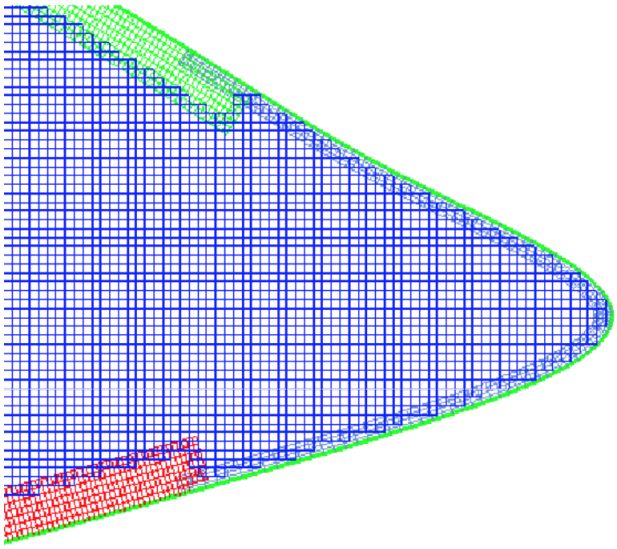


Figure 5.2: Example of overlapping finite difference grids using Overture [48].

were used in the discretization. Through this new model, it was observed that blinking has an effect on the cooling of the tear film [18, 19]. Driscoll and Braun then extended these techniques to simulate parabolic flow on an eye-shaped domain [25]. They studied a second-order parabolic PDE as a simplified model problem for the tear film by finding numerical solutions for the model

$$h_t + \nabla \cdot \mathbf{q}(h, h_x, h_y) = 0, \quad (x, y) \in \mathcal{E}(t), \tag{5.1}$$

where h is the film thickness and $\mathbf{q}$ is a known flux function. The domain $\mathcal{E}(t)$ changes with time, as visualized in Figure 5.3 and defined in the following section. In particular they examined a second-order thin film analog where

$$\mathbf{q} = -(A - Bh^{-3})\nabla h \tag{5.2}$$

for constants A and B [25]. Numerically the PDE is discretized with a Chebyshev polynomial tensor product approximation. In the case of the second order thin film analog with flux (5.2), this led to a solution that was accurate to at least five digits.

In this chapter, we seek to extend this method to solve a fourth order tear film model in a DAE in which the pressure is found from an algebraic equation. In order to

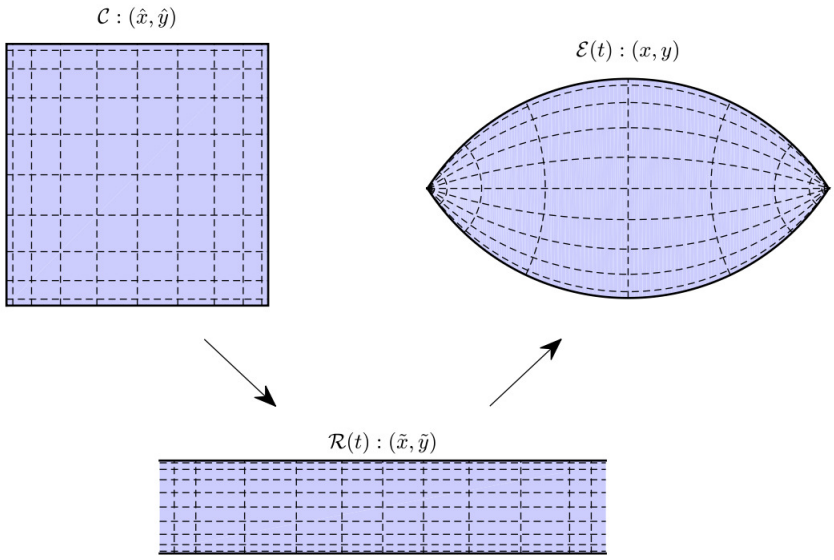


Figure 5.3: Plot of the subdomains used in [25], where $\mathcal{C}$, $\mathcal{R}(t)$, $\mathcal{E}(t)$ domains are the square, infinite strip and eye-shaped domain. The mappings between the domains are explained in the following section.

meet the computational demands we will use the preconditioned techniques of Chapter 3 to solve this problem. In the following sections, we describe the fourth order tear film model, as well as briefly review how the blinking eye domain $\mathcal{E}(t)$ is transformed from the $[-1, 1]^2$. We conclude with a numerical study of our method on the PDE.

5.2 Model description

We numerically solve for the height of the tear film $h(x, y, t)$ and pressure $p(x, y, t)$ with the model (5.1). Using flux $\boldsymbol{q} = \frac{h^3}{3}\nabla p$ where the pressure is found from an algebraic equation, this produces the model

$$\begin{aligned} h_t &= -\nabla \cdot \left(\frac{h^3}{3} \nabla p \right), \\ p &= -S\nabla^2 h - Ah^{-3}, \end{aligned} \qquad (x, y) \in \mathcal{E}(t) \tag{5.3}$$

where S represents the ratio of surface tension to viscous forces while A is the ratio of van der Waals to viscous forces [13]. On the boundary we enforce a Dirichlet condition $h = h_0$ as well as flux conditions on $\boldsymbol{q}$,

$$(n \cdot \boldsymbol{q})|_{\partial\mathcal{E}(t)} = Q_{\text{in}}(x, y, t) + Q_{\text{out}}(x, y, t) \tag{5.4}$$

for prescribed influx and enflux functions Q_{in}, Q_{out} that incorporate the moving boundary [45, 39, 13]. This condition ensures that the total volume of fluid over on $\mathcal{E}(t)$ is conserved over a blink cycle.

The domain $\mathcal{E}(t)$ is determined by a pair of two-dimensional transforms. First a fixed square $\mathcal{C} = [-1, 1]^2$ is mapped to an infinite strip $\mathcal{R}(t)$ by

$$\mathcal{R}(t) = \{(x, y) : |y| < \lambda(t)\}, \tag{5.5}$$

where $-1 < \lambda(t) < 1$ is a prescribed function. We use the realistic blink motion from [19]. The domain $\mathcal{C}$ is mapped to $\mathcal{R}(t)$ via

$$g_x(\hat{x}) = \frac{\gamma \hat{x}}{\alpha^2 - \hat{x}}, \quad g_y(\hat{y}) = \frac{1}{2}(\hat{y} + 1)(\lambda(t) + 1) - 1, \quad (\hat{x}, \hat{y}) \in \mathcal{C} \tag{5.6}$$

for some $\gamma > 0$ and $\alpha \geq 1$.

The domain $\mathcal{E}(t)$ is the image of $\mathcal{R}(t)$ under the conformal map

$$f(z) = \tanh(z), \tag{5.7}$$

i.e.,

$$\mathcal{E}(t) = \{\text{Re}(f(x + iy)), \text{Im}(f(x + iy))|(x, y) \in \mathcal{R}(t)\}. \tag{5.8}$$

If we define $\hat{\nabla}$ to be the gradient in terms of $(\hat{x}, \hat{y})$ then

$$\hat{\nabla} = \frac{1}{|f'(g_x(\hat{x}) + ig_y(\hat{x}))|} \left(\frac{(\hat{x}^2 + \alpha^2)\gamma}{(\hat{x}^2 - \alpha^2)^2} \frac{d}{d\hat{x}}, \frac{2}{1 + \lambda(t)} \frac{d}{d\hat{y}} \right). \tag{5.9}$$

Defining $\hat{h}(\hat{x}, \hat{y}) = h(x, y)$, $\hat{p}(\hat{x}, \hat{y}) = p(x, y)$, we can determine the tear-film hight and pressure by solving the following index-1 DAE on $\mathcal{C}$:

$$\begin{aligned} \hat{h}_t &= -\frac{1}{3}\hat{\nabla} \cdot (\hat{h}^3 \hat{\nabla} \hat{p}) - \frac{\dot{\lambda}(t)(1 + \hat{y})}{\lambda(t) + 1} \hat{h}_{\hat{y}} \\ \hat{p} &= -S\hat{\nabla}^2 \hat{h} - A\hat{h}^{-3}. \end{aligned} \tag{5.10}$$

The normal conditions of (5.4) greatly simply on $\mathcal{C}$. The influx function Q_{in} transforms to

$$Q_{\text{in}}(\hat{x}, \hat{y}, t) = \begin{cases} Q_{lg}(\hat{x}, t) + \dot{\lambda}(t)|f'(g_x(\hat{x}) + ig_y(\hat{x}))|(h_0 - h_e/2) & \hat{y} = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{5.11}$$

Figure 5.4: Plot of subdomains used for blinking eye model.

where h_e is the thickness of tear fluid under eyelid [39], and Q_{lg} is the prescribed influx from the lacrimal gland [13]. For the outflux function we have

$$Q_{\text{out}}(\hat{x}, \hat{y}, t) = \begin{cases} Q_p(\hat{y}, t) & \hat{x} = -1 \\ 0 & \text{otherwise,} \end{cases} \tag{5.12}$$

where $Q_p(\hat{y})$ describes the efflux out to the puncta [13]. The flux functions Q_{lg} and Q_p are defined in more detail within Appendix B.

5.3 Numerical experiments

For our first experiment, we choose A and S to be 6.11×10^{-6} and 3.09×10^{-6}, values taken from [13]. The blink cycle L is approximately 5.28. The thickness of the tear film under the lid h_e is set to 2 and $V_d = 1, V_s = 1$. In all experiments, $\alpha = 1.4$, $\gamma = 7$. We use the initial condition described in Appendix B, forcing the volume at $t = 0$ to be 24.

We solve the model using SNK to solve for the backward time step within ode15s (as described within section 4.4), using the partitioning in Figure 5.4 to better adapt to the features of the tear film height. For each subdomain we use a 35×35 Chebyshev interpolant approximation.

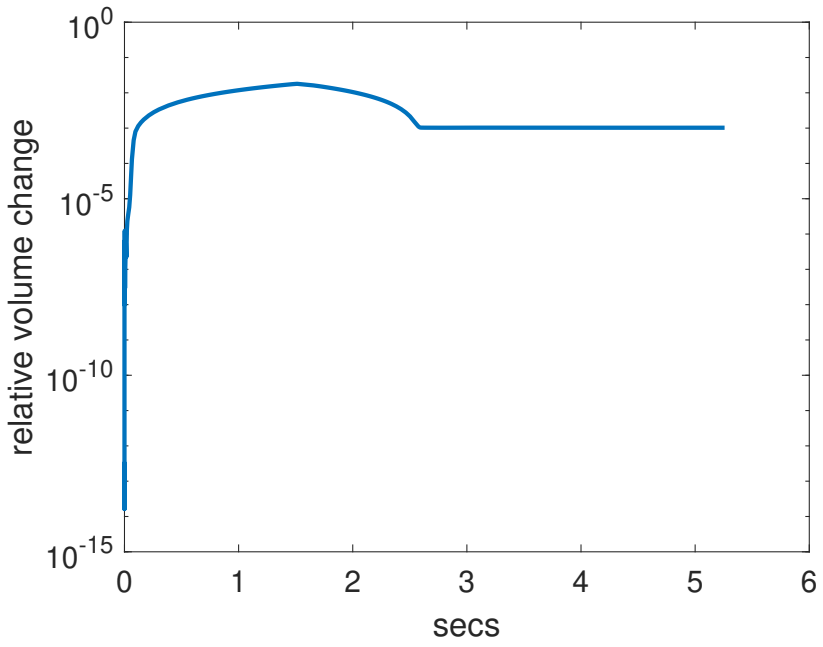
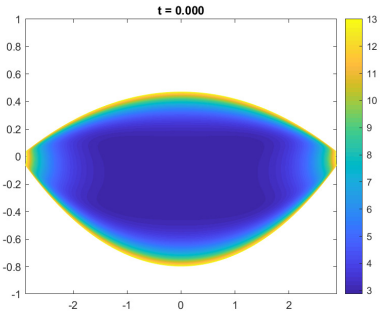


Figure 5.5: Change of volume of the fluid over time for a stationary boundary. The boundary conditions ensure constant volume over time.

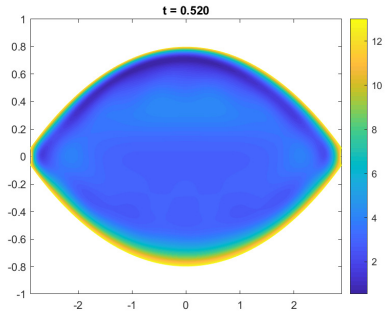
We first examine the case of a stationary domain. The boundary conditions of 5.3 ensure that the volume of the fluid $h(x, y, t)$ is constant over time (in the case of a stationary boundary). As seen in Figure 5.5, our method maintains a volume that is relatively within 10^{-3} of the initial volume, which is indicative of the pointwise error of the numerical solution.

We now solve the model with the same parameters but with a moving boundary prescribed by the realistic blink motion of [19] such that the eye closes 20 percent. With the moving boundary, we expect the volume to be conserved over a blink cycle (that is, it is conserved periodically). After one blink cycle the volume is conserved within a relative difference of 5×10^{-3}. While the solution was positive throughout the blink cycle, it does become close to zero on the upper left part of the lid $t = 1$, as seen in Figures 5.6-5.7.

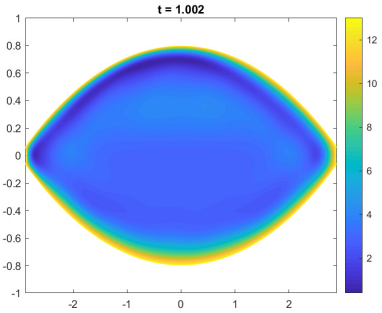
This issue becomes more prominent as V_s, V_d increase. As a last experiment we examine the model under more realistic conditions, where the eye closes to 70 percent



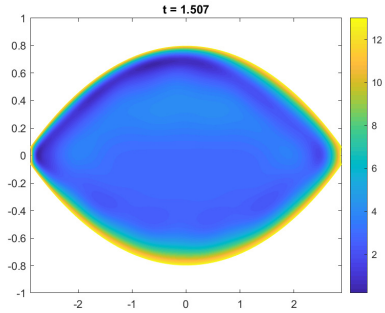
(a) Plot at t=0



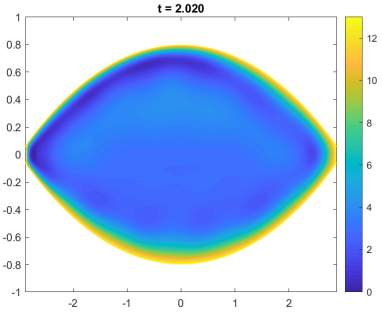
(b) Plot at t=0.520



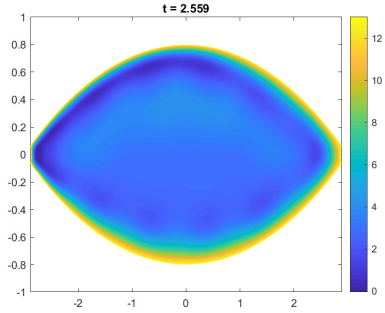
(c) Plot at t=1.002



(d) Plot at t=1.507

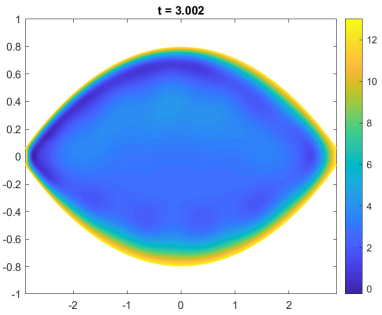


(e) Plot at t=2.020

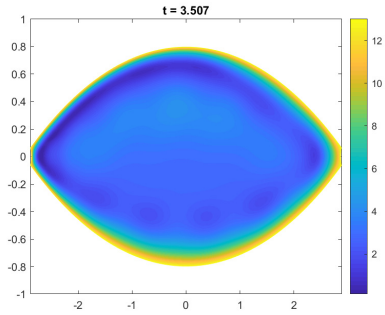


(f) Plot at t=2.559

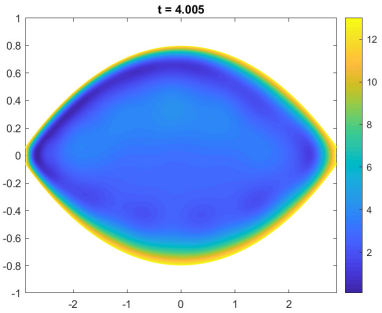
Figure 5.6: Plot of the tear film height for times between 0 and 2.559, where the total influx and outflux is set to 1 and the eye closes 20 percent



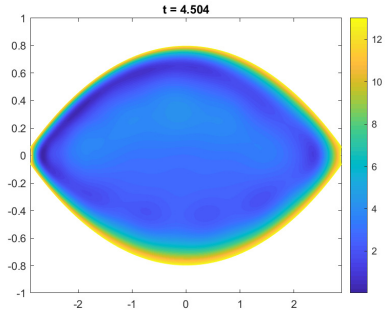
(a) Plot at t=3.002



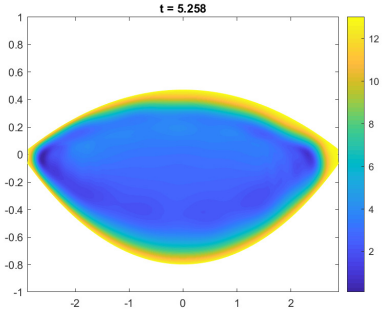
(b) Plot at t=3.507



(c) Plot at t=4.005

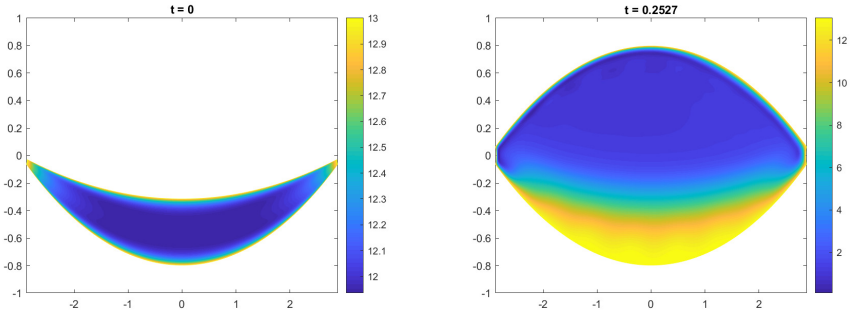


(d) Plot at t=4.504



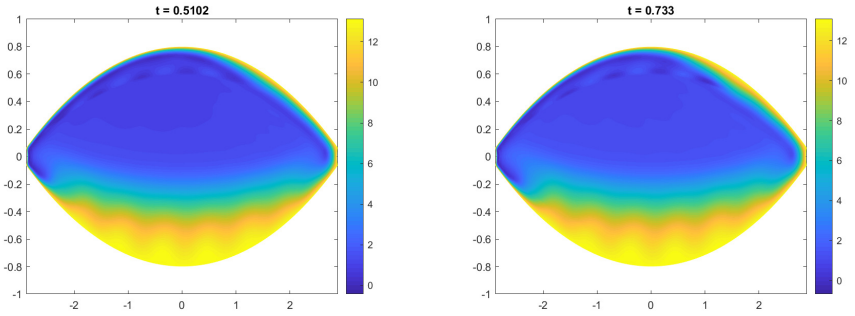
(e) Plot at t=5.258

Figure 5.7: Plot of the tear film height for times between 3.002 and 5.258, where the total influx and outflux is set to 1 and the eye closes 20 percent



(a) Plot at t=0.

(b) Plot at t=0.2527.



(c) Plot at t=0.5102

(d) Plot at t=0.733

Figure 5.8: Plot of the tear film hight, where the total influx and outflux is set to 4 and the eye closes 70 percent.

and $V_s = 4, V_d = 4$. Numerically we found that this model fails at around 0.7 seconds, as seen in Figure 5.8 since the tear film height becomes negative near the upper left corner. We suspect that this is because the outflux condition (5.12) is overpowering the influx condition (5.11). This could likely be mitigated by relaxing the bump function (B.3) for the puncta fluid efflux, i.e. reducing the height and increasing the width. The solution could also be under-resolved, since there are oscillations in the tear film height starting at t=0.5, as seen in Figure 5.7d-5.7e.

Chapter 6

CONCLUSION

In Chapter 2 we created a new method that offers a simple way to adaptively construct infinitely smooth approximations of functions that are given explicitly or that solve BVPs. We created a new way to implicitly construct a partition of unity recursively with a binary tree, allowing us to avoid the need to determine neighbors of each patch. This makes using a partition of unity scheme effective within a adaptive scheme; for example, when adaptively solving a BVP we can avoid having to redetermine neighboring patches with our new technique.

We extend the tree based approximations to arbitrary dimension in Chapter 3, allowing us to construct approximations that are adapted to the features of the function but still infinitely smooth. We utilized the tree structure to develop fast algorithms for integration, differentiation and interpolation. The tree structure gives a fast way to arithmetically combine partition of unity approximations by merging their splittings; it is not clear how this could be done with a partition of unity approximation represented with a graph (i.e., a data structure where a patch points to its neighbors). We extended this technique to arbitrary domains via the least squares technique.

In Chapter 4 we described a framework for overlapping domain decomposition in which overlap regions are discretized independently by the local subdomains, even in the formulation of the global problem. Communication between subdomains occurs only via interpolation of values to interface points. This formulation makes it straightforward to apply high-order or spectral discretization methods in the subdomains and to adaptively refine them. The technique may be applied to precondition a linearized PDE, but it may also be used to precondition the nonlinear problem before linearization, to get what we call the Schwarz–Newton–Krylov (SNK) technique. In doing so,

one gets the same benefit of faster Krylov inner convergence, but the resulting nonlinear problem is demonstrably easier to solve in terms of outer iterations and robustness.

We apply SNK to solve a blinking eye model in Chapter 5. Domain decomposition allows us to achieve the necessary resolution to perserve the volume over a blink cycle in a way that is computationally efficient. We discovered through numerical experiments that the influx and outflux conditions (5.11,5.12) could be causing the tear film thickness to become negative, producing unrealistic results. Further examination and experimentation is needed to validate the model.

The partition of unity techniques we presented in Chapter 3 allow for the possibility of parallelization. In particular our methods for adaptive construction could be sped up, since no communication is needed between subdomains in further refinement (i.e. once a domain is split, no communication is needed in between the resulting subdomains in further refinements). We explored how to parallelize our new SNK method using `parfor` loop in MATLAB, and found that it can offer some potential speed benefits. With SNK, the only communication between subdomains that is necessary is through the interface conditions. Work should be done to explore a technique that can take advantage of this (since `parfor` does not).

BIBLIOGRAPHY

[1] Ben Adcock and Daan Huybrechs. On the resolution power of Fourier extensions for oscillatory functions. *Journal of Computational and Applied Mathematics*, 260:312–336, 2014.

[2] Ben Adcock, Daan Huybrechs, and Jesús Martín-Vaquero. On the numerical stability of Fourier extensions. *Foundations of Computational Mathematics*, 14(4):635–687, 2014.

[3] Kevin W. Aiton and Tobin A. Driscoll. An adaptive partition of unity method for Chebyshev polynomial interpolation. *SIAM Journal on Scientific Computing*, 40(1):A251–A265, jan 2018.

[4] Kevin W. Aiton and Tobin A. Driscoll. An adaptive partition of unity method for multivariate Chebyshev polynomial approximations. *SIAM J. Sci. Comput.*, in revision.

[5] Jared L. Aurentz and Lloyd N. Trefethen. Chopping a Chebyshev series. *ACM Trans. Math. Softw.*, 43(4):33:1–33:21, January 2017.

[6] J. Bäck, F. Nobile, L. Tamellini, and R. Tempone. Stochastic spectral Galerkin and collocation methods for PDEs with random coefficients: a numerical comparison. In J.S. Hesthaven and E.M. Ronquist, editors, *Spectral and High Order Methods for Partial Differential Equations*, volume 76 of *Lecture Notes in Computational Science and Engineering*, pages 43–62. Springer, 2011. Selected papers from the ICOSAHOM '09 conference, June 22-26, Trondheim, Norway.

[7] Zachary Battles. *Numerical linear algebra for continuous functions*. PhD thesis, University of Oxford, 2005.

[8] Zachary Battles and Lloyd N Trefethen. An extension of MATLAB to continuous functions and operators. *SIAM J. Sci. Comp.*, 25(5):1743–1770, 2004.

[9] Asgeir Birkisson. *Numerical solution of nonlinear boundary value problems for ordinary differential equations in the continuous framework*. PhD thesis, Oxford University, UK, 2013.

[10] Asgeir Birkisson and Tobin A Driscoll. Automatic fréchet differentiation for the numerical solution of boundary-value problems. *ACM Transactions on Mathematical Software (TOMS)*, 38(4):26, 2012.

[11] O Botella and R Peyret. Benchmark spectral results on the lid-driven cavity flow. *Computers & Fluids*, 27(4):421–433, 1998.

[12] Achi Brandt and Oren E. Livne. *Multigrid Techniques*. Society for Industrial and Applied Mathematics, jan 2011.

[13] Richard J Braun, P Ewen King-Smith, CG Begley, Longfei Li, and NR Gewecke. Dynamics and function of the tear film in relation to the blink cycle. *Progress in retinal and eye research*, 45:132–164, 2015.

[14] Anthony J Bron, Mark B Abelson, George Ousler, E Pearce, Alan Tomlinson, Norihiko Yokoi, Janine A Smith, Carolyn Begley, Barbara Caffery, Kelly Nichols, et al. Methodologies to diagnose and monitor dry eye disease: report of the diagnostic methodology subcommittee of the international dry eye workshop (2007). *Ocular Surface*, 5(2):108–152, 2007.

[15] Xiao-Chuan Cai and David E. Keyes. Nonlinearly preconditioned inexact Newton algorithms. *SIAM Journal on Scientific Computing*, 24(1):183–200, jan 2002.

[16] Xiao-Chuan Cai and Marcus Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(2):792–797, jan 1999.

[17] G Chesshire and William D Henshaw. Composite overlapping meshes for the solution of partial differential equations. *Journal of Computational Physics*, 90(1):1–64, 1990.

[18] Quan Deng, Richard J Braun, TA Driscoll, and P Ewen King-Smith. A model for the tear film and ocular surface temperature for partial blinks. *Interfacial phenomena and heat transfer*, 1(4), 2013.

[19] Quan Deng, RJ Braun, and Tobin A Driscoll. Heat transfer and tear film dynamics over multiple blink cycles. *Physics of Fluids*, 26(7):071901, 2014.

[20] J. E. Dennis and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1987.

[21] V. Dolean, M. J. Gander, W. Kheriji, F. Kwok, and R. Masson. Nonlinear preconditioning: How to use a nonlinear Schwarz method to precondition Newton's method. *SIAM Journal on Scientific Computing*, 38(6):A3357–A3380, jan 2016.

[22] Victorita Dolean, Pierre Jolivet, and Frédéric Nataf. *An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation*. Society for Industrial and Applied Mathematics, 2015.

[23] T. A. Driscoll, N. Hale, and L. N. Trefethen, editors. *Chebfun Guide*. Pafnuty Publications, 2014.

[24] Tobin A Driscoll, Folkmar Bornemann, and Lloyd N Trefethen. The chebop system for automatic solution of differential equations. *BIT Numerical Mathematics*, 48(4):701–723, 2008.

[25] Tobin A Driscoll, Richard J Braun, and Joseph K Brosch. Simulation of parabolic flow on an eye-shaped domain with moving boundary. *Journal of Engineering Mathematics*, 111(1):111–126, 2018.

[26] Tobin A Driscoll and Nicholas Hale. Rectangular spectral collocation. *IMA Journal of Numerical Analysis*, 36(1):108–132, 2015.

[27] Tobin A Driscoll and JAC Weideman. Optimal domain splitting for interpolation by Chebyshev polynomials. *SIAM Journal on Numerical Analysis*, 52(4):1913–1927, 2014.

[28] Stanley C. Eisenstat and Homer F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM Journal on Scientific Computing*, 17(1):16–32, jan 1996.

[29] Bengt Fornberg and Natasha Flyer. *A primer on radial basis functions with applications to the geosciences*. SIAM, 2015.

[30] Richard Franke. A critical comparison of some methods for interpolation of scattered data. Technical report, Naval Postgraduate School, Monterey, California, 1979.

[31] Richard Franke and Greg Nielson. Smooth interpolation of large sets of scattered data. *Internat. J. Numer. Methods Engrg.*, 15(11):1691–1704, 1980.

[32] Martin J. Gander. Schwarz methods over the course of time. *Electronic Transactions on Numerical Analysis*, 31:228–255, 2008.

[33] Alan Genz. A package for testing multiple integration subroutines. In *Numerical Integration*, pages 337–340. Springer, 1987.

[34] Ilene K Gipson. Distribution of mucins at the ocular surface. *Experimental eye research*, 78(3):379–388, 2004.

[35] Michael Griebel and Marc Alexander Schweitzer. A particle-partition of unity method for the solution of elliptic, parabolic, and hyperbolic PDEs. *SIAM J. Sci. Comp.*, 22(3):853–890, 2000.

[36] Nicholas Hale and Lloyd N Trefethen. Chebfun and numerical quadrature. *Science China Mathematics*, 55(9):1749–1760, 2012.

[37] Behnam Hashemi and Lloyd N. Trefethen. Chebfun in three dimensions. *SIAM Journal on Scientific Computing*, 39(5):C341–C363, jan 2017.

[38] William D Henshaw. Ogen: An overlapping grid generator for overture. *LANL unclassified report*, pages 96–3466, 1998.

[39] A Heryudono, RJ Braun, TA Driscoll, KL Maki, LP Cook, and PE King-Smith. Single-equation models for the tear film in a blink cycle: realistic lid motion. *Mathematical medicine and biology: a journal of the IMA*, 24(4):347–377, 2007.

[40] Charles Hirsch. *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*. Elsevier, 2007.

[41] FJ Holly and MA Lemp. Tear physiology and dry eyes. *Survey of ophthalmology*, 22(2):69–87, 1977.

[42] Daan Huybrechs. On the Fourier extension of nonperiodic functions. *SIAM Journal on Numerical Analysis*, 47(6):4326–4355, 2010.

[43] Michael E Johnson and Paul J Murphy. Changes in the tear film and ocular surface from dry eye syndrome. *Progress in retinal and eye research*, 23(4):449–474, 2004.

[44] Andreas Klimke and Barbara Wohlmuth. Algorithm 847: Spinterp: piecewise multilinear hierarchical sparse grid interpolation in MATLAB. *ACM Transactions on Mathematical Software*, 31(4):561–579, dec 2005.

[45] Longfei Li, Richard J Braun, Tobin A Driscoll, William D Henshaw, Jeffrey W Banks, and P Ewen King-Smith. Computed tear film and osmolarity dynamics on an eye-shaped domain. *Mathematical medicine and biology: a journal of the IMA*, 33(2):123–157, 2015.

[46] Longfei Li and RJ Braun. A model for the human tear film with heating from within the eye. *Physics of Fluids*, 24(6):062103, 2012.

[47] Longfei Li, RJ Braun, KL Maki, WD Henshaw, and Peter Ewen King-Smith. Tear film dynamics with evaporation, wetting, and time-dependent flux boundary condition on an eye-shaped domain. *Physics of Fluids*, 26(5):052101, 2014.

[48] Kara L Maki, Richard J Braun, William D Henshaw, and P Ewen King-Smith. Tear film dynamics on an eye-shaped domain i: pressure boundary conditions. *Mathematical medicine and biology: a journal of the IMA*, 27(3):227–254, 2010.

[49] KL Maki, RJ Braun, TA Driscoll, and PE King-Smith. An overset grid method for the study of reflex tearing. *Mathematical medicine and biology: a journal of the IMA*, 25(3):187–214, 2008.

[50] KL Maki, RJ Braun, P Ucciferro, WD Henshaw, and PE King-Smith. Tear film dynamics on an eye-shaped domain. part 2. flux boundary conditions. *Journal of Fluid Mechanics*, 647:361–390, 2010.

[51] John C Mason and David C Handscomb. *Chebyshev Polynomials*. CRC Press, 2002.

[52] Roel Matthysen and Daan Huybrechs. Function approximation on arbitrary domains using Fourier extension frames. *arXiv preprint arXiv:1706.04848*, 2017.

[53] Dermot H McLain. Two dimensional interpolation from random data. *The Computer Journal*, 19(2):178–181, 1976.

[54] S Mishima and DM Maurice. The oily layer of the tear film and evaporation from the corneal surface. *Experimental eye research*, 1(1):39–45, 1961.

[55] J Daniel Nelson, Jun Shimazaki, Jose M Benitez-del Castillo, Jennifer P Craig, James P McCulley, Seika Den, and Gary N Foulks. The international workshop on meibomian gland dysfunction: report of the definition and classification subcommittee. *Investigative ophthalmology & visual science*, 52(4):1930–1937, 2011.

[56] MS Norn. Semiquantitative interference study of fatty layer of precorneal film. *Acta ophthalmologica*, 57(5):766–774, 1979.

[57] Ricardo Pachón, Rodrigo B Platte, and Lloyd N Trefethen. Piecewise-smooth chebfuns. *IMA journal of Numerical Analysis*, 30(4):898–916, 2010.

[58] Rodrigo B Platte and Lloyd N Trefethen. Chebfun: a new kind of numerical computing. In *Progress in industrial mathematics at ECMI 2008*, pages 69–87. Springer, 2010.

[59] Luis G Reyna and Michael J Ward. On the exponentially slow motion of a viscous shock. *Communications on Pure and Applied Mathematics*, 48(2):79–120, 1995.

[60] Ali Safdari-Vaighani, Alfa Heryudono, and Elisabeth Larsson. A radial basis function partition of unity collocation method for convection–diffusion equations arising in financial applications. *J. Sci. Comp.*, 64(2):341–367, 2015.

[61] Lawrence F Shampine and Mark W Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.

[62] Jie Shen. Numerical simulation of the regularized driven cavity flows at high Reynolds numbers. *Computer Methods in Applied Mechanics and Engineering*, 80(1-3):273–280, 1990.

[63] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM, 1968.

[64] Barry Smith, Peter Bjørstad, and William Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 2004.

[65] Ireneusz Tobor, Patrick Reuter, and Christophe Schlick. Reconstructing multi-scale variational partition of unity implicit surfaces with attributes. *Graphical Models*, 68(1):25–41, 2006.

[66] A. Townsend and L. N. Trefethen. Continuous analogues of matrix factorizations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2173):20140585–20140585, nov 2014.

[67] Alex Townsend and Lloyd N Trefethen. An extension of Chebfun to two dimensions. *SIAM Journal on Scientific Computing*, 35(6):C495–C518, 2013.

[68] Lloyd N Trefethen. *Spectral Methods in MATLAB*, volume 10. SIAM, 2000.

[69] Lloyd N. Trefethen. Computing numerically with functions instead of numbers. *Communications of the ACM*, 58(10):91–97, sep 2015.

[70] Lloyd N Trefethen. Computing numerically with functions instead of numbers. *Communications of the ACM*, 58(10):91–97, 2015.

[71] Lloyd N Trefethen. Cubature, approximation, and isotropy in the hypercube. *SIAM Review*, 59(3):469–491, 2017.

[72] Lyod N. Trefethen. *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, 2013.

[73] Holger Wendland. *Scattered Data Approximation*. Cambridge University Press, 2004.

[74] Lan Zhong. *Dynamics and imaging for lipid-layer-driven tear film breakup (TBU)*. PhD thesis, University of Delaware, 2018.

Appendix A

MERGING TREES

Algorithm 14 describes a recursive method for merging two trees $\mathcal{T}_1$ and $\mathcal{T}_2$, representing functions f_1 and f_2, into a tree representation for $f_1 \circ f_2$, with $\circ$ as $+$, $-$, $\times$, or $\div$. The input arguments to the algorithm are the operation, corresponding nodes of $\mathcal{T}_1$, $\mathcal{T}_2$, and the merged tree, and the number r, which is the dimension that was most recently split in the merged tree. Initially the algorithm is called with root nodes representing the entire original domain, and $r = 0$.

We assume an important relationship among the input nodes. Suppose that $\mathsf{zone}(\nu_k)=\prod_{j=1}^{d}[\alpha_{kj}, \beta_{kj}]$ for $k = 1, 2$, and that $\mathsf{zone}(\nu_{\text{merge}})=\prod_{j=1}^{d}[A_j, B_j]$. Then we require for $k = 1, 2$ that

$$[a_{kj}, b_{kj}] = [A_j, B_j] \quad \text{for all } j \text{ having} \quad \mathsf{isdone}(\nu_k)_j = \text{FALSE}. \tag{A.1}$$

This is trivially true at the root level. The significance of this requirement is that it allows us to avoid ambiguity about what the zone of ν_{merge} should be after a new split in, say, dimension j. Since only an uncompleted dimension can be split, the zone of the children of ν_{merge} after splitting will be identical to that of whichever (or both) of the ν_k requires refinement in dimension j.

For example, suppose the zones of ν_1 and ν_2 are $[-1, 0] \times [-1, 1]$ and $[-1, 1] \times [0, 1]$, respectively, and $\mathsf{zone}(\nu_{\text{merge}})=[-1, 0] \times [0, 1]$. It is clear that we can interpolate from ν_1 and ν_2 onto ν_{merge}. It is also clear that we can further split in x in ν_1, and in y in ν_2. But if we were to split ν_2 in x, one of the children would have zone $[0, 1] \times [0, 1]$, which is inaccessible to ν_1.

Consider the general recursive call. If both ν_1 and ν_2 are leaves, then we simply evaluate the result of operating on their interpolants to get the values on ν_{merge}. If

exactly one of ν_1 and ν_2 is a leaf, then we split ν_{merge} the same way as the non-leaf and recurse into the resulting children; property (A.1) trivially remains true in these calls. If both ν_1 and ν_2 are non-leaves, and they both split in the same dimension, then we can split ν_{merge} in that dimension and recurse, and the zones will continue to match as in (A.1).

The only remaining case is that ν_1 and ν_2 are each split, but in different dimensions. In this case we have to use information about how the splittings are constructed in Algorithm 6. Recall that each unresolved dimension is split in order, while resolved dimensions are flagged as finished in all descendants. By inductive assumption, ν_{merge} was most recently split in dimension r. The algorithm determines which ν_k has splitting dimension j that comes the soonest after r (computed cyclically). Thus for all dimensions between r and j, neither of the given nodes splits, so it and its descendants all must have isdone set to TRUE in those dimensions, and property (A.1) makes no requirement. Furthermore, the dimension r_k does satisfy (A.1) for ν_k, and the same will be true for its children and the children of ν_{merge}. All other dimensions will inherit (A.1) from the parents.

Algorithm 14 merge($\circ$,ν_1,ν_2,ν_{merge},r)

if ν_1 and ν_2 are leaves **then**
  $\mathsf{values}(\nu_{\text{merge}}) := \mathsf{interpolant}(\nu_1) \circ \mathsf{interpolant}(\nu_2)$, evaluated on $\mathsf{grid}(T_{\text{merge}})$
else if ν_1 is a leaf **and** ν_2 is not a leaf **then**
  split(ν_{merge},$\mathsf{splitdim}(\nu_1)$)
  merge($\circ$,ν_1,$\mathsf{child}_0(\nu_2)$,$\mathsf{child}_0(\nu_{\text{merge}})$,$\mathsf{splitdim}(\nu_2)$)
  merge($\circ$,ν_1,$\mathsf{child}_1(\nu_2)$,$\mathsf{child}_1(\nu_{\text{merge}})$,$\mathsf{splitdim}(\nu_2)$)
else if ν_1 is not a leaf **and** ν_2 is a leaf **then**
  split(ν_{merge},$\mathsf{splitdim}(\nu_1)$)
  merge($\mathsf{child}_0(\nu_1)$,ν_2,$\mathsf{child}_0(\nu_{\text{merge}})$,$\mathsf{splitdim}(\nu_1)$)
  merge($\mathsf{child}_1(\nu_1)$,ν_2,$\mathsf{child}_1(\nu_{\text{merge}})$,$\mathsf{splitdim}(\nu_1)$)
else
  if $\mathsf{splitdim}(\nu_1)$=$\mathsf{splitdim}(\nu_2)$ **then**
    split(ν_{merge},$\mathsf{splitdim}(\nu_1)$)
    merge($\circ$,$\mathsf{child}_0(\nu_1)$,$\mathsf{child}_0(\nu_2)$,$\mathsf{child}_0(\nu_{\text{merge}})$,$\mathsf{splitdim}(\nu_1)$)
    merge($\circ$,$\mathsf{child}_1(\nu_1)$,$\mathsf{child}_1(\nu_2)$,$\mathsf{child}_1(\nu_{\text{merge}})$,$\mathsf{splitdim}(\nu_1)$)
  else
    $r_1 = (\mathsf{splitdim}(\nu_1) - r - 1) \mod d$
    $r_2 = (\mathsf{splitdim}(\nu_2) - r - 1) \mod d$
    If $r_1 > r_2$, swap ν_1 and ν_2
    split(ν_{merge},$\mathsf{splitdim}(\nu_1)$)
    merge($\circ$,$\mathsf{child}_0(\nu_1)$,ν_2,$\mathsf{child}_0(\nu_{\text{merge}})$,$\mathsf{splitdim}(\nu_1)$)
    merge($\circ$,$\mathsf{child}_1(\nu_1)$,ν_2,$\mathsf{child}_1(\nu_{\text{merge}})$, $\mathsf{splitdim}(\nu_1)$)
  end if
end if

Appendix B

FLUX FUNCTIONS AND INITIAL CONDITION

We define the influx and outflux functions Q_{lg}, Q_p to be bump functions in space and time. With

$$\text{step}(x, a, b) = 1/2(1 + \tanh[(4(a + b - 2x))/(a - b)]), \tag{B.1}$$

we define a indicator like bump function

$$w(x, a, b, l) = \text{step}(x, a, a + l) - \text{step}(x, b - l, b) \tag{B.2}$$

that has support on $[a, b]$ but continuously drops from 1 to 0 in intervals of length l. Here w effectively is a smooth approximation of an indicator function.

With this bump function and a blink cycle with period L, we define

$$\hat{Q_{lg}}(\hat{x}, t) = w(t, 0.02, 0.5L, 0.1) \exp(((\hat{x} - 0.5)/0.25)^2), \tag{B.3}$$

$$\hat{Q_p}(\hat{y}, t) = w(t, 0.05, 0.5L, 0.1)w(\hat{y}, -0.9, 0.25, 0.9). \tag{B.4}$$

With target supply and drain volumes V_s, V_d over a blink cycle, we normalize the flux functions (B.6) to define Q_p, Q_{lg}:

$$Q_{lg}(\hat{x}, t) = \frac{V_s}{\int_0^p \int_{-1}^1 \hat{Q_{lg}} d\hat{x}dt} \hat{Q_{lg}}(\hat{x}, t), \tag{B.5}$$

$$Q_p(\hat{y}, t) = \frac{V_d}{\int_0^p \int_{-1}^1 \hat{Q_p} d\hat{y}dt} \hat{Q_p}(\hat{y}, t). \tag{B.6}$$

The influx function Q_{lg} is defined so there is an influx of fluid from the top of the lid, concentrated in the center; this is simulated with the use of an exponential bump function, as seen in (B.5). The influx occurs for $t \in [0.02, 0.5L]$. The ouflux function Q_p defines a drainage through the puncta; Q_p is defined on the left corner of the eye-shaped domain. The outflux occurs evenly throughout the corner, as seen with the use

of the indicator like function (B.2) in (B.6). The outflux occurs within $t \in [0.05, 0.5L]$, starting after an initial influx of fluid. A plot of the flux functions at different times throughout the blink cycle can be seen in Figure B.1.

We define the initial condition $h_I(\hat{x}, \hat{y})$ implicitly by first solving

$$\begin{aligned} \nabla^2 \hat{h_I} &= 10(\hat{x}^4 + \hat{y}^4) \quad (\hat{x}, \hat{y}) \in \mathcal{C} \\ \hat{h_I} &= 1 \text{ on } \partial\mathcal{C}, \end{aligned} \tag{B.7}$$

and finding coefficients c_0, c_1 so that the initial condition

$$h_I(\hat{x}, \hat{y}) = c_1 \hat{h_I}(\hat{x}, \hat{y}) + c_0 \tag{B.8}$$

matches h_0 on the boundary and numerically has a prescribed volume V_{init}. This initial condition produces a solution with boundary layers, as seen in Figures 5.6a,5.8c.

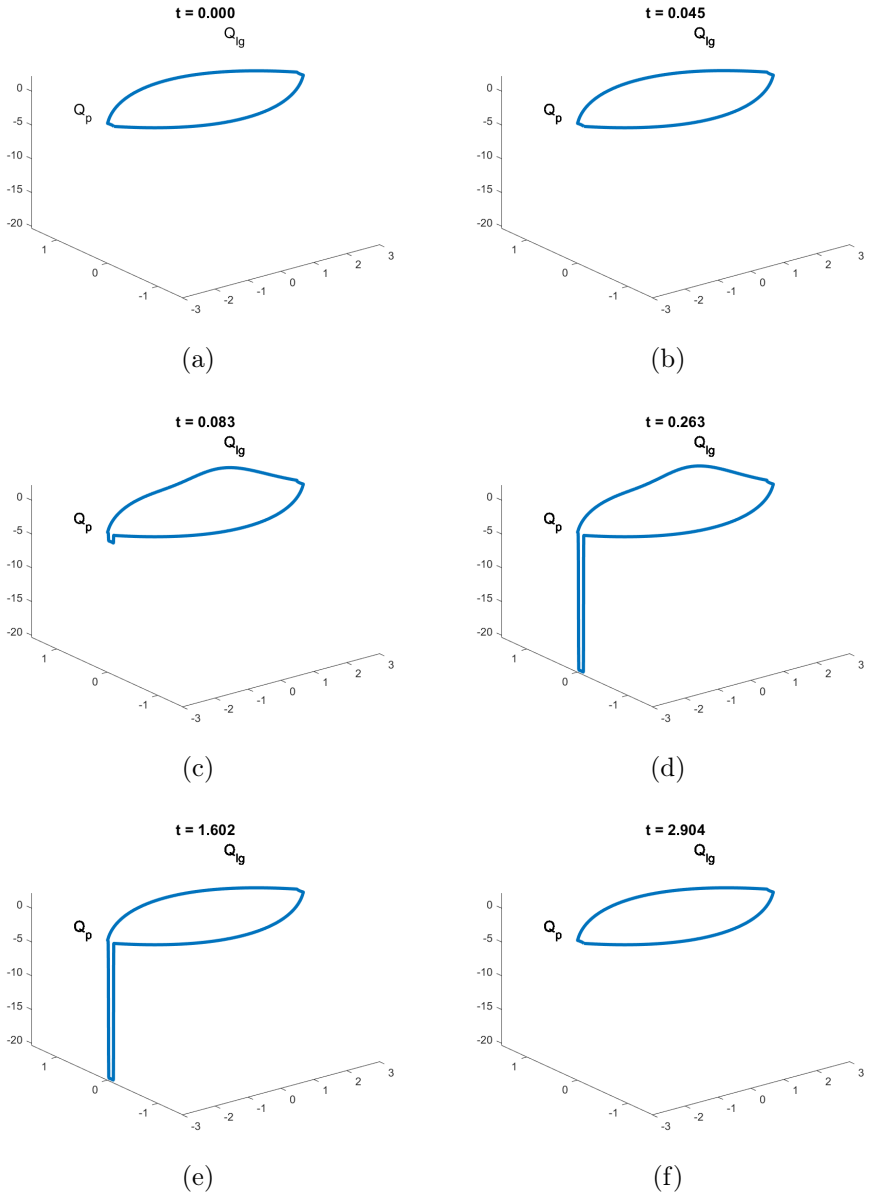


Figure B.1: Plot of the flux functions (B.5,B.6) at different times throughout the blink cycle. At the start of the blink cycle both fluxes are 0. Then at $t = 0.02$, an influx of fluid starts, followed by an outflux at $t = 0.05$. From $t = 0.05$ to $t = 0.5L$, there is both an influx and outflux of fluid. After $t = 0.5L$, both the influx and outflux of fluid dissipates.