SOLVING LAPLACE PROBLEMS WITH CORNER SINGULARITIES

by

Caelen Feller

Supervisor: Professor Kirk M. Soodhalter

Submitted to Trinity College Dublin School of Maths

March 2020

Abstract

In this project, I aim to further existing research in the use of rational function approximations to solve Laplace equations in two dimensions with corner singularities. My aim is to allow solutions to handle more complicated boundary conditions and domains than is currently possible. This requires accounting for discontinuous and non-Dirichlet boundary conditions, or computationally inconvenient situations such as elongated domains and non-convex domains. I implement and extend the recently published algorithms of Gopal and Trefethen, placing their work in the context of approximation theory literature, and benchmark and discuss the performance of this implementation against industry standard methods such as the finite and boundary element methods.

Introduction

Approximation theory is the study of how functions can be approximated by simpler functions which converge to the true function as they get more complex. In particular, it is concerned with the errors of these approximations - how far from correct they are. There are many ways to define error in this case over some set, typically depending on the norm of the space in question, and whether an average or maximal error is sought. A commonly sought-after concept in approximation theory is an optimal approximation, i.e. that no better error can be achieved using simple functions of that type. A commonly known example are Taylor polynomials, which can be used to approximate holomorphic functions arbitrarily accurately. In fact, polynomials can be used to approximate holomorphic functions with uniform convergence on certain domains, according to the Runge approximation theorem. This sort of knowledge of the way the approximations converge is highly useful, as it gives us optimised, rapidly converging computer implementations of these approximation schemes, allowing for complicated functions to be represented programmatically using simple, efficient ones such as polynomials.

This project focuses on rational function approximations, which were established as an indispensable object of approximation theory in the early 20th century following the work of D.J. Newman on the absolute value function and the resulting attention given to their theory. As outlined in [23, Chapter 23], a rational function of *type (m,n)* (analogous to the order of a polynomial) is a quotient of polynomials $\frac{p}{q}$, $p \in \mathcal{P}_m$, $q \in \mathcal{P}_n$, where $\mathcal{P}_n = \{\text{polynomials of degree at most } n\}$. The set of rational functions of type (m,n) is denoted as $\mathcal{R}_{mn}$, and $\mathcal{R}_n := \mathcal{R}_{nn}$. For a more useful characterisation, let the numerator and denominator polynomials be relatively prime, take the degree of numerator and denominator to be exactly m and n respectively, and let the leading coefficient of q be 1. Then the rational function p/q is said to have exact type (m,n). 0 is defined to have exact type $(-\infty, 0)$. One can represent rational functions in several ways, either as the direct quotient of the polynomials or as partial fractions, as in [9, Theorem 4.4h].

It should be noted that while a well-established tool, rational functions are not always an improvement over polynomial approximations. As discussed in [23], there is a very simple example showing this - given an analytic function on an ellipse, there is actually no gain in the order of convergence with respect to the number of parameters (if you use type (n, n) rational functions, a typical approach). They are not a catch-all, and the determination of the use cases where they are an improvement is a focus of Stenger in particular in [11].

The Laplace equation, in case the reader is unfamiliar, is a differential equation where

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0$$

I will consider it in 2 dimensions, in Cartesian/complex space only in this project. It is one of simplest differential equations, with functions solutions known as harmonic functions. It is simple to imagine what a solution to this equation looks like on a domain in the complex plane, considering the smoothing effect this condition has. As with all partial differential equations, we must consider it in the context of a boundary value problem. Namely, we search for a solution given some known values on the boundaries of the domain, either of the solution itself (Dirichlet boundaries) or its derivatives (Neumann boundaries).

In this project I seek to apply the rational functions above to approximation of solutions to the Laplace problem on polygonal domains, and extend these results to the cases mentioned previously of new boundary conditions and computationally challenging domains, which are explained in more detail later. It can be shown that such approximations converge faster than most methods, with what is known as root-exponential convergence. This means that as the order grows (the index of approximating function if they are considered as a series), we can see the error decreasing as an exponentially in the root of this order. In contrast, uniform convergence only guarantees linearity.

This report is broken into four main chapters. I first discuss the background of the theory I use in order to give historical context to the research and methods used. I then discuss the theory introduced by Gopal & Trefethen and potential extensions to it, in order to give rigour to my implementation. The next section discussed the algorithm described in their paper, with additional detail on my implementation, which is done in MATLAB and presented in Appendix A. Finally, I evaluate the performance of this implementation, identifying where it functions well and comparing it to other standard methods of solving PDEs. I subsequently present some final recommendations on the project.

Acknowledgements

I would like to deeply thank all those who helped me complete this project and my final year of study, in particular my supervisor, Professor Kirk M. Soodhalter and the faculty of the School of Maths in TCD for offering whatever support and assistance I needed, even in the midst of a global crisis, my friends for remaining patient with me throughout the process, and my family for supporting me at every turn. I would also like to thank Professor Trefethen and Abinand Gopal for authoring such an inspirational paper.

TABLE OF CONTENTS

Chapter 1

Background

In this chapter I examine the results of approximation theory which led to the publication of Gopal & Trefethen's 2019 paper, [21]. They provide useful context to the results of their paper and my project. Below is a summary of notable advancements/areas of research in this space, in chronological order.

Reverse Chronological List of Movements in the Field:

1. **Initial results on prototypical functions(1970→1980)** - Functions which are fundamental in developing the theory are explored initially by Newman and Gončar.

2. **Sharpening of bounds, algorithms (1980→2010)** - The estimates previously made are refined further, exact values found by Stahl and Stenger. Newer work considers potential algorithms this is applicable to.

3. **Multiple dimensions and algorithms/implementation (2010→Now)** - Use of least squares for fitting the approximations, consideration of SVD(Singular Value Decomposition) for interpolation, and application to PDEs/conformal mappings and other.

In the coming sections I will discuss why they are noteworthy, presenting necessary results, influence on other research, and their reception by the academic community. I keep my explanations of these results relatively brief where possible as their role is to provide context for the accompanying definitions and results, and an overview of the history of the field.

1.1 Early Developments

I start with D.J. Newman's 1964 paper, which was a seminal example in approximation theory of a proof that rational approximations offer convergence improvements over polynomial approximations. His results show that rational approximations converge more quickly than polynomials for $|x|$, which is described as "a prototype function for functions with singularities" by [23]. He shows the best order of convergence for this function with rational approximation is *root-exponential*, i.e. the error of the approximation is of the order $O(e^{-C\sqrt{n}})$, where n is the order of approximation and C a positive constant. One can follow research citing Newman's results here all the way to the present day, and thus chart the development of this theory.

Newman was motivated to explore the approximation of $|x|$ due to connections to fundamental results of complex analysis, and the proposal of the problem by H.S. Shapiro. These include the fact that the polynomial approximation of $|x|$ is used in Lebesgue's proof of the Weierstrass Approximation theorem and a proof of Jackson's Theorem, see [1, 7]. Additionally, such a result was given motivation as it was shown by Bernstein in 1912 that degree n polynomial approximations give at best linear ($O(n^{-1})$) convergence [2], which inspired desire for more rapid convergence.

Formally, Newman's result is as follows. Define the *order* of a rational approximations as the maximum of the degrees of the numerator and denominator. I denote the most accurate approximation relative to the supremum norm on a segment (also known as the minimax error) $\Delta \subseteq \mathbb{R}$ of a function $f : \Delta \to \mathbb{C}$ by rational functions of order no higher[1] than n as:

$$R_n(f, \Delta) = \inf_{r_n \in \mathcal{R}_n} \sup_{x \in \Delta} |f(x) - r_n(x)| \tag{1.1}$$

In this notation, the primary result of Newman's paper can be written as the bound Equation 1.2, with the rational approximation scheme which yields this result bound given in Equation 1.3. Sharpening of the bounds was accomplished by Stahl in [15], as discussed in section 1.2.

$$e^{-C_1\sqrt{n}} < R_n(|x|, [-1, 1]) < e^{-C_2\sqrt{n}}, \; n \geq 1 \tag{1.2}$$

$$p(x) = \prod_{k=0}^{n-1} (x + \exp(-n^{-1/2})), \; r(x) = x\frac{p(x) - p(-x)}{p(x) + p(-x)} \tag{1.3}$$

In Figure 1.1, the convergence bounds of Equation 1.2 are demonstrated experimentally, yielding an ultimately straight line graph when plotted logarithmically against the square root of the order. Also note that in this scheme, poles are exponentially clustered about the origin in $\mathbb{C}$, a common feature of rational approximations discussed in this report which will be discussed further later.

Newman's result was surprising to theorists, given the linear behaviour of polynomial approximations for $|x|$. Within ten years of its publication, some thirty publications reference it, with more than three hundred 56 years later. Notably, A. A. Gončar published several papers following on from Newman, some discussing the results in terms of potential theory and the work of Zolotarev, that I will discuss later. The main advancement of Gončar was

[1]Note: Though this is only specified for type (n, n) functions, it gives us the best rational approximation order of convergence as type (n, n) rational approximations include type (m, k), $m, k \leq n$.

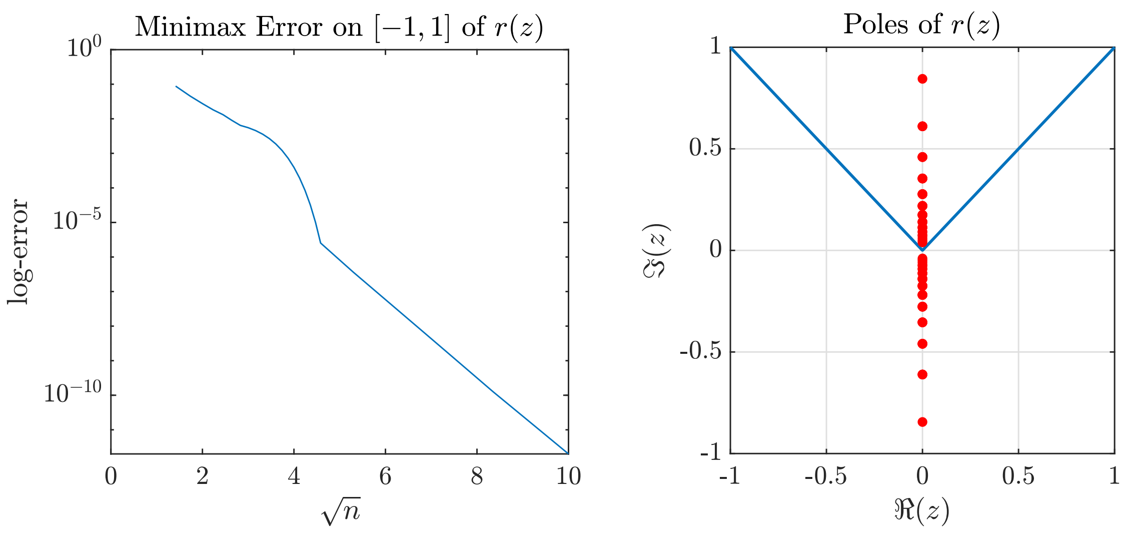

Figure 1.1 Convergence and Exponentially Clustered Poles of Newman Scheme

the extension of the theory to any function continuous on $[0, 1]$ with a bounded analytic continuation on a circle about a real point including $(0, 1]$. This notably applies to functions with characteristic singularities on such as $|x|$.

He establishes bounds above and below for rational approximation order of such functions in [3][4], bounding $R_n(f_\phi, [-1, 1])$, such that

$$f_\phi(x) = \begin{cases} 0, & x \in [-1, 0] \\ \phi(x), & x \in [0, 1] \end{cases}$$

He then extends this form to the more general f defined above. It is in this way that you can treat $|x|$ as a prototype for singular functions. Several lemmas are established to prove that if f is continuous on $[0, 1]$ and coincides on the interval $(0, 1]$ with a function bounded and analytic on $\Delta_1(-1)$, then $R_n(f, [0, 1]) = O(\rho_n)$, $\rho_n = \inf_{1\leq t\leq\infty} \left[te^{-Cn/t} + \omega(e^{-t})\right]$, where ω is the modulus of continuity of f. As such, for f bounded and analytic on $D = \Delta_1(-1)$, then you get the following bound on the order of convergence.

$$R_n(f, [h, 1]) < C \sup_{z\in D} |f(z)| \ \log\left(\frac{1}{h}\right) \exp\left(-\frac{Cn}{\ln 1/h}\right) \tag{1.4}$$

The modulus of continuity of a function is a function taken such that all points in the domain obey $|f(x) - f(y)| \leq \omega(|x - y|)$.

The other characteristic function where rational approximations perform well is that of e^x on $(-\infty, 0]$, which as noted previously is representative of unbounded domains. In Cody, Meinardus and Varga [6], it was proposed to attempt this. Here, polynomial approximations are no competition at all when considered over the whole domain, as they will diverge to ∞ as $x \to -\infty$ unless constant. As noted in their paper, it is only necessary to consider the inverse of the Taylor expansion truncation for e^x on $[0, \infty)$ here to find a lower bound of geometric convergence $O(2^{-n})$ on convergence, strengthened later to $O(3^{-n})$ by Schönhage in [8] using general inverse polynomial approximations. Later, Newman shows the upper limit on rational approximation accuracy is geometric for this function in [10].

1.2 Sharp Bounds and Algorithms

While the developments considered previously were largely theoretical, we come to a period of theory where the algorithms and computational methods involved are considered further. In his 1983 paper Stenger asked whether it is possible to tell a priori when it is expected that the Thiele algorithm or the ϵ -algorithm produce an accurate rational approximation, and how these algorithms compare with the convergence of the best possible approximation for certain classes of analytic functions on intervals [11]. I will define them fully shortly, but the Thiele/ϵ algorithms were the leading ways of constructing rational approximations when interpolating between points at the time. Note that is not until the 2000's that higher dimensions were considered, largely due to computational constraints.

Many papers at this time were concerned with extending and sharpening the bounds provided by Newman as Gončar did previously or that not all classes of functions are better approximated with rational functions, generally expanding the theory of the area, see [23, Chapter 23–26] for further discussion of these. Notably, it was shown by Szabados in [5] that for approximations on $[-1, 1]$ of functions analytic on the ellipse E_R they would perform worse than polynomial approximations. Here E_R is the ellipse with foci at $z = \pm 1$, and sum of semi-axes $= R$.

In this paper, Stenger uses when possible the notation and context of SINC approximations, as these were a large research interest for him at the time. SINC methods involve approximating using expansions of the form in Equation 1.5, and share some properties with rational approximation methods as a non-linear approximation scheme according to Stenger.

$$C_{M,N}(f,h)(x) = \sum_{k=-M}^{N} f(kh)\,\mathrm{sinc}\left(\frac{x}{h} - k\right),\ \ \mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \tag{1.5}$$

The similarities he states are that the optimal interpolation points to choose and the types of functions they approximate better than polynomials are similar for low degree rational functions. While in some cases SINC methods are more theoretically useful, they are less easily handled computationally as noted by Stenger.

The rational approximations given by Stenger have several computationally useful properties. Firstly, poles of approximations in this paper are on the real line outside of interval (if finite or semi-finite). This is one of the first papers considering pole positioning as done later, as in the setting of the paper poles are entirely pre-determined. Secondly, the rational functions are linear in the function being approximated, which makes them more easily manipulated. Finally, Stenger describes them as near-optimal, as they satisfy a tight bound on the error described in Equation 1.6. The main theorem of the paper states that for functions on the Hardy Space H_p for $p \in (1,\infty)$ multiplied by $1/(1-z^2)$, the order of convergence satisfies this bound with $A, B, N > 0$

$$AN^{-1}\exp\left(-\pi\frac{2N(p-1)}{p}\right) \leq R_{2N+2,2N+1} \leq BN^{1/2}\exp\left(-\pi\left(\frac{N(p-1)}{2p}\right)^{1/2}\right) \tag{1.6}$$

The paper then discusses the implications for leading approximation algorithms at the time. These are worth discussing here as once and potentially future alternatives to the use of least-squares in this report for calculation of rational approximations.

The Thiele Algorithm The Thiele Algorithm (or ρ-algorithm) constructs rational approximations by representing the function as a continued fraction. Given a set of interpolation points $\{x_0, \ldots, x_m\}$, I construct intermediary functions

$$\rho_0^j = f(x_j),\ j = 0, 1, \ldots, m$$
$$\rho_1^j = \frac{x_{j+1} - x_j}{\rho_0^{j+1} - \rho_0^j},\ j = 0, 1, \ldots, m$$
$$\rho_i^j = \frac{x_{i+j} - x_j}{\rho_{i-1}^{j+1} - \rho_{i-1}^j} + \rho_{i-2}^{j+1},\ j = 0, 1, \ldots, m,\ i = 2, 3, \ldots, m$$

$$r(x) = \rho_0^0 + \frac{x - x_0}{\rho_1^0} + \frac{x - x_1}{\rho_2^0 - \rho_0^0} + \cdots + \frac{x - x_{m-1}}{\rho_m^0 - \rho_{m-2}^0} \tag{1.7}$$

These intermediate functions yield a rational function interpolating between the given points, $r \in R_{m/2}$, specified in Equation 1.7. A noteworthy feature is that the prediction of the algorithm on unbounded domains will be well-defined as $\lim_{x\to\infty} r(x) = \rho_m^0$, though this is not initially obvious.

The Epsilon Algorithm Given a sequence of numbers $S_j, j = 0, \ldots, m$, typically a series of sums, similar to the Thiele algorithm, define ϵ_i^j, intermediary functions by

$$\epsilon_0^j = S_j,\ j = 0, 1, \ldots, m$$
$$\epsilon_1^j = \frac{1}{\epsilon_0^{j+1} - \epsilon_0^j},\ j = 0, 1, \ldots, m$$
$$\epsilon_i^j = \frac{1}{\epsilon_{i-1}^{j+1} - \epsilon_{i-1}^j} + \epsilon_{i-2}^{j+1},\ j = 0, 1, \ldots, m,\ i = 2, 3, \ldots, m$$

You can use this algorithm to evaluate the Padé approximation, which is the rational approximation which matches the target function as closely as possible in Taylor expansions at a given point. To do so, let S_j be the following, and the Padé approximation is given by ϵ_{2k}^n, where c_i are the coefficients of the Taylor expansion of f.

$$S_j = \sum_{i=0}^{j} c_i\tau^i,\ \epsilon_{2k}^n = \frac{p_{n+k-1}(\tau)}{q_k(\tau)}$$

There is more to discuss on Padé approximations, but this is not an optimal algorithm for finding them. Using SVD, you can establish a better algorithm, as discussed in [23, Chapter 27].

In 1992, Herbert Stahl published the final entry in a long line in a series of results designed to sharpen Newman's bound on the error for $|x|$. He showed that $R_nn(|x|) = 8e^{-\pi\sqrt{n}}$ exactly in [13]. For $|x|^\alpha$, the results are expanded due the above theorems. In 2003, Stahl took the $|x|$ prototype one step further, by considering the approximation of $|x|^\alpha$ in [15]. This can be used to consider functions which satisfy the Hölder condition of order α. This is similar to Lipshitz continuity, stating that $\|f(x) - f(y)\| \le M \|x - y\|^\alpha, M > 0$. He considers its approximation on the interval $[0, 1]$, though transferring the result to other intervals is shown to work in the paper. The paper's main result is the following asymptotic behaviour

$$\lim_{n\to\infty} e^{2\pi\sqrt{\alpha n}} R_{nn}(x^\alpha, [0,1]) = 4^{1+\alpha} |\sin(\pi\alpha)|, \ \forall\alpha > 0$$

with corollary

$$\lim_{n\to\infty} e^{2\pi\sqrt{\alpha n}} R_{nn}(|x|^\alpha, [-1,1]) = 4^{1+\alpha/2} \left|\sin\left(\frac{1}{2}\pi\alpha\right)\right|, \ \forall\alpha > 0$$

This marks an end in theory designed to generalise Newman's results directly, by proving sharper bounds or bounds for other functions, as at this point the two main prototypical functions were considered, and other cases in one dimension followed from these as discussed in [23].

1.3 Recent Developments

After the theoretical developments of the previous century, we are left with two main considerations. The first is the application of preceding results to multiple dimensions. The second is the algorithm used to find these approximations. In [21], Gopal and Trefethen propose a solution to the Laplace problem in two dimensions using the "Least Squares" method, a commonly used optimisation method. They show this approach will have root-exponential convergence, that this is the optimal rational approximation method possible on a convex, simply connected domain with corners, and discuss ways in which this result can be made more general. This paper was the inspiration for this project, and I will go through the theory in detail in the next chapter, but will discuss the paper's inspirations and impact here.

Firstly, how do we come to using least squares, rather than the highly specialised algorithms of the previous section? In considering the applications of rational approximations to PDEs, I move from approximation of a function to interpolation between certain points. What is meant by this is that rather than try and find the "best" rational approximation to a total function, I seek to find the "best" approximation to it for a set of points. What is meant by “best” here varies depending on the setting, but mostly refers to the supremum norm of error at these points.

The first complication in constructing a rational interpolation is presented by Trefethen in [23], where it is known that if a rational function takes the same value at distinct points, it must be constant, and as such not all interpolation problems can be satisfied. As two values to be interpolated approach each other in value, a pole is introduced between the two points. This is referred to as an “extraneous pole” in the literature and is frequently undesirable. To avoid this, I seek to optimise the error, which will frequently lead to solutions which are not exact, but can avoid localised poles in the domain of solution. Note also that for the main method of this paper, the poles are pre-assigned, so this is less important. It is still desirable that the optimal placement of poles would not include internal poles however, as we wish for a smooth approximation to the solution within the bulk.

The use of Least Squares in this area was previously explored by Trefethen, Pachón, Gonnet, Deun, and Güttel in [18, 19, 17]. In [18], they consider the problem of interpolation to values on a set of pre-determined points by rational functions (the Cauchy interpolation problem). They initially consider the problem on roots of unity, with rational approximations of order (N, N), which are equivalent to the number of roots of unity. This is nearly equivalent to the problem of finding the following expression.

$$p(z_j) = q(z_j)f(z_j),\ j = 0, \dots, N,\ z_j = e^{2\pi ij/(N+1)},\ \text{with } p, q \text{ polynomials}$$

This problem does not have unique solution unless we normalise q, letting its coefficient vector have standard euclidean norm of 1. It then will satisfy the original rational problem at z_j if we have $q(z_j) \neq 0$, as otherwise an extraneous pole will be introduced. We can then consider the p-coefficients as a linear function of the q-coefficients. Finding a solution to this problem reduces to an application of SVD to find the null space of this function, reduced by the twice the rank of the function so that the problem is well-conditioned in machine arithmetic. Further details are described in [17]. I will discuss the SVD further briefly later, but as a standard numerical linear algebra method its exact details are unimportant here.

The use of least squares comes into play when we consider the case of more interpolation points than the order of our approximation. In such a scenario, an exact solution is not possible. Note that when minimising error, care must be taken as extraneous poles within the domain (causing infinite error), can go undetected by sampling points chosen. As mentioned, the approach taken here involves fixing the poles in advance, and computing the optimal coefficients based on these fixed poles.

The next question is why these results on rational approximations are now being applied to PDE problems, which have not been mentioned up to this point, and why to the Laplace equation in particular. It also raises a potential issue - as the results are constrained to special classes of PDEs in two dimensions, how useful are these methods computationally? The Laplace equation is of particular interest as its solutions will be harmonic functions, which correspond to the real and imaginary parts of holomorphic functions. This connection to complex analysis enables the proofs of Trefethen to take place, making use of results such as the Runge approximation theorem and the Hermite integral formula. Other methods of computation, discussed in section 4.2, handle this problem well but have issues near corners in the domains with precision. The Laplace problem is seen a prototype for PDEs, with some of the simplest analytic solutions in many cases. With it, it is possible to attack problems like the Poisson equation, which is an in-homogenous version of the Laplace problem, or more complex equations such as Hemholtz equations. Gopal and Trefethen claim they are currently working on the Hemholtz case in the final remarks of [21], and discuss loose ideas for it in [20].

In the year since the paper's release, reference code was released by Gopal and Trefethen[2]. This not only provided a useful comparison point for my own implementation of their algorithm, but also seems to have inspired the numerical analysis community to make use of the results further. Of note are two preprints, [26, 25]. In the first, Costa has approached the Laplace problem suing the AAA (Adaptive Antoulas–Anderson) algorithm. This is a general algorithm for rational interpolation which is a more sophisticated version of the one using SVD mentioned previously. This gives slower solution to polygonal laplace problems, but is a more general algorithm and can be used to approach problems in conformal mappings, as discussed in this paper and by Trefethen in [24]. An interesting sub-case considered here is the inverse of the domains I consider, where we solve over the entire complex plane excluding a polygonal domain. In the first, Hoskins and Rachh work on developing a solution to the case with Neumann boundaries, something which I approach in subsection 3.2.3 in a simpler fashion. The method they discuss involves considering the corners in a more local sense.

A question I have not addressed to this point is that of motivation. Why do we care about rational approximation methods? They have been used to approach a variety of problems before, to varying effect and adoption. Trefethen compiles a list of such applications in [23, Chapter 23]. I have highlighted some of these and other applications here.

1. *Engineering Signal Filters.* In order to computationally represent filters such as band-pass filters, we need to consider waveforms with discontinuous jumps. Rational approximations offer a natural solution for handling this on some disk, while polynomials will struggle to capture the structure quickly as noted by Trefethen.

2. *Convergence acceleration.* Often it is desirable in the sciences to extrapolate a non-analytical approximate solution to the limit of some sequence known up to a point, $\lim_{k\to\infty} a_k$. If considered as the limit of the reciprocal functions $\lim_{h\to 0} a_{1/h}$, we can apply the ϵ-algorithm previously mentioned to find a solution.

3. *Holomorphic continuation.* Computing the holomorphic continuation of a function across a larger region than typically computable is done using Richardson extrapolation based on rational approximations in [23, Chapter 28].

[2]Implementation available at https://people.maths.ox.ac.uk/trefethen/lightning.html

4. *Matrix exponentiation.* The Padé algorithm is used to compute a rational approximation to the exponential function of a matrix, which is noted as being the most efficient method by Trefethen, citing Ward.

5. *Conformal Mappings* See Trefethen's paper on conformal mapping applications [24]. In this paper, clustering about corners is seen to be a common feature of optimal or near-optimal rational approximations for conformal maps.

6. *Numerical PDE Solutions.* Much like the one I consider for Laplace problems, rational approximations can be applied to the Heat equation as one wishes to compute a matrix exponential in order to solve this.

Many other applications exist, but these were some of the more interesting ones, in my opinion, highlighting use in several different disciplines and even professions - from the complex analyst to a signal processing engineer to the physicist, rational approximations play a role.

Chapter 2

Theory

2.1 Theoretical Developments of Gopal & Trefethen

In this section, I will discuss the details of relevant results from the Gopal & Trefethen paper [21], which aimed to establish and explore the effectiveness of rational approximations for solving Laplace problems on domains with corners, primarily polygonal ones. The main result of this paper, as mentioned previously, is that the optimal rational approximation scheme will yield root-exponential convergence on a convex polygonal domain. To prove this, several preliminary results are introduced, the Hermite integral formula from complex analysis, the Runge approximation formula, and a version of theorem locally near the corners of the domain.

The Hermite integral formula they specify is presented for rational interpolation, with proof cited from Walsh. In summary, for a holomorphic function f on a simply connected domain Ω, with holomorphic continuation to the boundary, if we are provided with interpolation points on the domain and pre-assigned poles, $\{\alpha_k\}_{k=0}^{n-1}$ and $\{\beta_k\}_{k=0}^{n-1}$ respectively, then the we know its difference from the unique rational function, r, with these simple poles interpolating f. This difference is the following expression, where $\phi(z) = \prod_k (z-\alpha_k)/\prod_k (z-\beta_k)$.

$$f(z) - r(z) = \frac{1}{2\pi i} \int_{\partial\Omega} \frac{\phi(z)}{\phi(t)} \frac{f(t)}{t-z} dt \tag{2.1}$$

The Runge approximation theorem is a well-known result in complex analysis. Generally it states that for a compact $K \subset \mathbb{C}$, a holomorphic function on a open set about K will have uniformly convergent rational approximations, with poles in a specified set as long as the set contains a point from every connected component outside K. Its corollary is used in Gopal & Trefethen's proof — that if K is simply connected we get no poles, instead there exists a uniformly convergent set of polynomial approximations.

Gopal & Trefethen's local result states that for a bounded holomorphic function on the slit disk which satisfies $f(z) = O(|z|^\delta)$ as $z \to 0$, for some $\delta > 0$, then we can get type $(n-1, n)$ rational functions which converge root-exponentially on a convex sector of the disk about the positive real line, with poles clustered exponentially along the negative real line. The proof of the local result relies on the fact that $|\phi(z)| \leq |z|$, where ϕ is as state in the Hermite integral formula. This does not hold in the case where we deal with a non-convex sector of the disk, i.e. where the internal angle of the sector exceeds π. Using this fact, they go on to bound the integrals yielded in the difference $f(z) - r(z)$ as defined in the Hermite integral formula, showing they are root-exponentially small for a sufficiently small disk about 0 in the sector. Their global result is the main theoretical result of the paper, and is stated here. The growth condition arises from the application of the local result.

Theorem 2.3. of [21] (Global root-exponential convergence) *Let Ω be a convex, polygonal domain with corners $\omega_1 \dots, \omega_m$, $f \in \mathcal{O}(\Omega)$, with holomorphic continuation to $\Delta_{\epsilon_k}(\omega_k) \setminus \{$the exterior bisector of domain at $\omega_k\}$.*

If $\exists\ \delta > 0$ s.t. $|f(z) - f(\omega_k)| = O(|z - \omega_k|^\delta)$, $z \to \omega_k \implies \exists$ degree n rational functions $\{r_i\}$, $i > 0$ such that $|f - r|_\Omega = O(\exp(-C\sqrt{n})), C > 0$, $n \to \infty$, with the finite poles of each r_n clustered exponentially along the exterior bisectors of the domain such that the number of poles near a corner grows in proportion to n, as $n \to \infty$

To prove this theorem, they break f down into two sums of Cauchy integrals of form,

$$\frac{1}{2\pi i}\int_G \frac{f(t)}{t-z}dt$$

each over different paths G. For half of them, G is the two sides of the exterior bisectors to the angles. For the other half, it is the curve joining the ends of the exterior bisector slits, adding up to the boundary of the closure of the region of holomorphicity. These paths are shown in Figure 2.1.

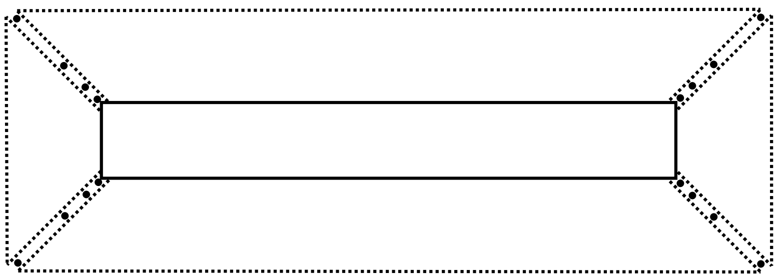

Figure 2.1 Paths about rectangular domain, with poles clustered along bisectors.

Aside from the obvious caveats of the hypotheses in the above theorem, as noted by Gopal & Trefethen in their additional remarks, the optimal approximation under a certain error norm is not always the best practically. While the supremum error norm is used above, a more useful norm may instead be a supremum norm where distances are weighted by distance to the corners of the domain, such as in section 2.3, or similar.

2.2 Extension to Harmonic functions

Previously in Theorem 2.3. of [21], we state the theorem for holomorphic functions. However, it is desirable to extend this to cover harmonic functions, as then we can apply it to the solutions of the Laplace problem, allowing us to guarantee the same convergence. This extension is discussed in [21] briefly, but with sufficient detail such that I can present a complete treatment of their sketch proof here. A harmonic function $u : \Omega \to \mathbb{R}$ can be defined as a real-valued function on a domain Ω, s.t. $u \in C^\infty(\Omega)$ and it satisfies the Laplace equation on that domain, $\Delta u = 0$

A standard result is that any harmonic function on Ω, which is simply-connected, is the real part of a holomorphic/analytic function $f(z) = u + iv$, uniquely determined up to an additive constant (see [12, Chapter 5, Section 6]). As such, we can approximate u using the approximation for f, the corresponding holomorphic function, with all results proceeding from Theorem 2.3. of [21], assuming we can satisfy the hypotheses.

Namely, given a u harmonic on Ω with harmonic extension across the boundaries and about the slits at each corner, similar to the holomorphic extension in the statement of the theorem, with $u(z) - u(\omega_k) = O(|z - \omega_k|^\delta)$ for some $\delta > 0$ and each k, we are able to find that v has the same properties and thus so does f.

We begin by using the Riemann mapping theorem to find a bi-holomorphic (conformal) mapping to the unit disk, $\phi : \Omega \to \Delta_0(1)$. As composition of a holomorphic function with a holomorphic function will yield a holomorphic function and harmonic functions are the real and imaginary components of holomorphic functions, we get that $\hat{u} = u \circ \phi^{-1}$ is harmonic, and that its harmonic conjugate is preserved, unique as usual up to an additive constant. Of note is that due to the continuity of ϕ, our growth condition is preserved.

$$u(z) - u(\omega_k) = O((z - \omega_k)^\delta) \implies \hat{u}(t) - \hat{u}(\phi(\omega_k)) = O(|t - \phi(\omega_k)|^\delta)$$

We can then apply the Hilbert transform, which takes $\hat{u} \mapsto H(\hat{u}) = \hat{v}$, the harmonic conjugate on the unit disk. As the Hilbert transform is bounded under the L_p-norm, for $1 < p < \infty$, we have the desired result, that this growth property applies to $\hat{v} = h(\hat{u})$. Using the continuity of ϕ we can get this same growth property on Ω and the extension of the theorem to harmonic functions is complete.

2.3 Discontinuous Extension

What is meant here is a case where the boundary conditions indicate a discontinuity in the solution at the corner or otherwise. An example would be all but one side of the domain having a value of 0, with the remaining side having a value of 1. Theoretically, the existence of solutions for the boundary value problem holds, though it is necessary to define what the value of the boundaries are at the corners themselves. The main consideration in extending the results to discontinuous boundary conditions is that we must consider what is now meant by convergence. We no longer can expect the supremum of the error on the boundary to converge to zero, due to the discontinuity. A potential option to resolve this is weighting the error as you approach the corner by some factor of the distance, as this will allow the overall error to converge to zero.

The other consideration is that of the error on the boundary presenting a bound on the error in the domain. This is the case if we are dealing with the weighted error, but with a slight modification, that we instead have the error is bounded by the precision over the distance to the nearest corner.

2.4 Non-Convex Extensions

The next most evident limitation to the theorem is the fact that the domain is non-convex. Gopal & Trefethen believe that this the theorem can be extended to this case. They state that the local theorem should hold when the disk segment is non-convex, and present a sketch of the beginning of a global argument using potential theory methods. In my efforts to extend their theory, I attempted to follow their suggestions here by seeing which assumptions of the local arguments failed. This did not yield a proof for me, and given how universally all the assumptions made early on fail, I think a dramatically different proof method would be necessary if it does hold. From my experimental results, I think it does hold for some simpler subset of functions without modification, where I saw purely root-exponential convergence.

For example, the assumption that $|z - \alpha_j| < |z - \beta_j|$, where $\beta_j = -e^{-\sigma_j/\sqrt{n}}$ and $\alpha_j = -\beta_j$ is the clustering scheme of the poles fails for any z in the left half of the disk, and in many cases this causes $|\phi(z)| > |z|$, which means we need an entirely new bounding method for the Hermite integrals, which I was not able to find.

As a demonstration of why this is awkward computationally, note the behaviour of the solution in the region of re-entrant corners in chapter 4. A *re-entrant* corner of a polygonal domain is one which "re-enters" the domain - i.e. at the corner $\theta > \pi$, where θ is the internal angle. Note the pair to this definition, that *salient* corners are those which have $\theta < \pi$. The convexity assumption of the domain is equivalent to assuming that all corners are salient). Intuitively, it causes the approximation scheme to have to account for boundary conditions within the bulk if a salient corner extends far into the domain.

The potential theory argument required a large portion of additional theory which I will not go into here as I believe it was beyond the scope of what was possible in a undergraduate FYP. They claim that allowing the interpolation points to distribute themselves such that they minimise an energy function based on a potential function of the logarithm of the product of the distance to the interpolation points and poles will be optimal. If this potential function can be shown to be smaller on the interior of the domain, then the root-exponential convergence should apparently hold on general domains, but there is insufficient detail for me to infer how this follows.

2.5 Other Extensions

These represent some extensions discussed without much detail by Gopal & Trefethen, which I have implemented. I discuss the main considerations and complications of the theory here.

Curved Boundary Segments

The previously specified theory works in the case where boundaries are not straight line segments but instead curves which do not affect the convexity of the domain. The implementation needs some modification, which is discussed in subsection 3.2.2. Obviously, we have complications when the segments cause non-convex areas of the domain, as we can then have areas looping around on themselves and behaving quite irregularly. Experimentally, it is sometimes possible to avoid poles within the domain by placing along some non-bisector line from the corner as in Figure 2.2. This can still converge, but tends to be slower as discussed in chapter 4.

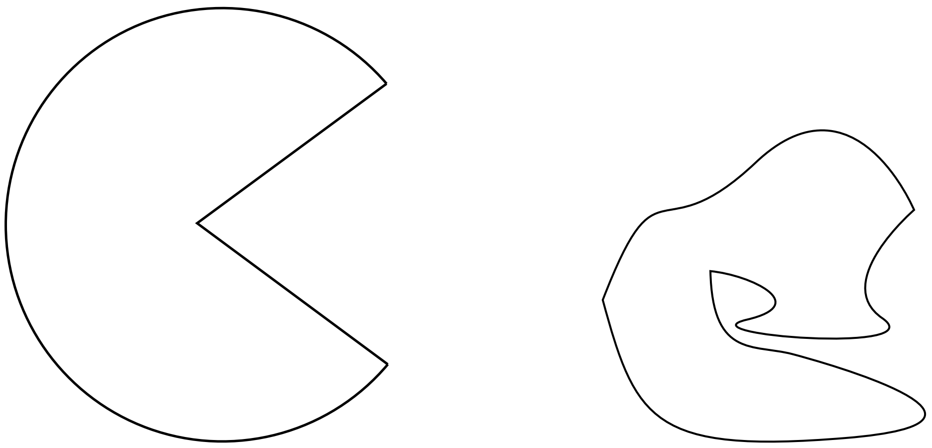

Figure 2.2 Normal Curved Domain, Bad Curved Domain

Neumann Boundary Conditions

With Neumann boundary conditions, the derivative of the solution at the boundary along the normal vector at that point is specified instead of a fixed value or function, which is a Dirichlet boundary condition. Once again, we do not have to change the theory to handle this case, unless the boundary conditions mandate discontinuity in the boundary, as the theory itself does not refer to the boundary conditions used to find the rational approximation. It does require a different approach to computation however, and a computationally accurate method of estimating the derivative of the rational approximation at the boundary.

Chapter 3

Algorithms

3.1 Trefethen & Gopal Algorithm

In this chapter, I discuss the algorithms making use of principles studied in the previous chapter, which I implement and test in MATLAB in the following chapter. As noted previously, I implement the algorithm desribed by Gopal and Trefethen in order to apply the results of Theorem 2.3. of [21]. In summary, this involves fitting Equation 3.1's coefficients with a least-squares solution against the boundary data.

$$r(z) = \overbrace{\sum_{j=1}^{N_1} \frac{a_j}{z - z_j}}^{\textit{NEWMAN}} + \overbrace{\sum_{j=0}^{N_2} b_j (z - z_*)^j}^{\textit{RUNGE}}. \tag{3.1}$$

Abstractly, the details are shown in algorithm 1. I will go into more technical detail subsequently, providing insights into decisions made due to technical constraints in my MATLAB implementation, for which annotated code is provided in Appendix A.

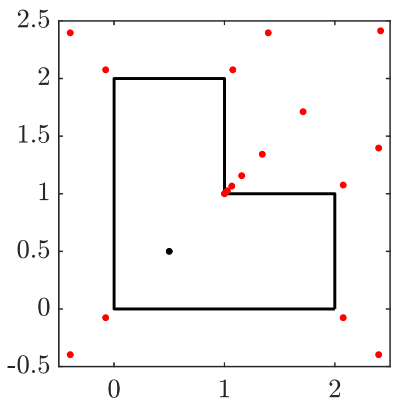

Figure 3.1 Example domain for algorithm 1, showing poles and central point

Data: $\omega_i \in \mathbb{C}, i = 1, \ldots, m$, the corners of the domain (blue points in Figure 3.1)
$h : \mathbb{C} \to \mathbb{R}$, the boundary conditions of the domain (only Dirichlet here)
$\epsilon \in \mathbb{R}_{>0}$, the desired error threshold, and $z_* \in D$, a central point of the domain
Result: $c \in \mathbb{R}$, the coefficients of Equation 3.1

```
1  $n = 3$                                   // Constant determining order of approximation
2  while $\|AC - B\|_\infty < \epsilon$ do
       // Determine $N_1 = O(mn)$ poles.  Here, $\hat{w_i}$ is the vertex normal at $\omega_i$
       // and $\beta_j = -\exp(-\sigma(\sqrt{n} - \sqrt{j}))$, $\sigma \approx 4$ is determined experimentally
3      for $i = 1$ to $m$ do
4          if $\omega_i$ re-entrant then  $p_{i,j} = \omega_i +,\ j = 1, \ldots, 3n$
5          else $p_{i,j} =,\ j = 1, \ldots, n$
6      end
7      Remove all $p_{i,j}$ which are inside of the domain
8      $N_2 = \lceil n/2 \rceil$                        // Degree of monomials to be determined
9      $N = 2N_1 + 2N_2 + 1$  // The total number of terms in the approximation
       // Find $M \approx 3N$ boundary sample points, clustered near the corners
10     for $p_{i,j},\ i = 1, \ldots, m,\ \forall j$, and $M = 1$ do
11         $\delta = \omega_i - p_{i,j}$              // Cluster boundary sample points like poles
12         $b_{M+l} = \omega_i + \delta / l(p_{i,j} - \omega i),\ l = 1, 2, 3$
13         $b_{M+3+l} = \omega_i - \delta / l(p_{i,j} - \omega i),\ l = 1, 2, 3$
14         if $b_{M+l},\ l = 1, \ldots, 6$ is outside domain then delete $b_{M+l}$
15         $M = M +$ number of added boundary points
16     end
       // Create $M \times N$ matrix $A$ in order to specify least-squares problem
17     $P_{i,j} = \left( \frac{1}{b_i - p_j} \right), \quad M_{i,j} = (b_i - w_*)^j$
18     $A_i = \left[ \{\Re(P_{i,j})\ \Im(P_{i,j})\}_{j=1,\ldots,N_1}\ \{\Re(M_{i,j})\ \Im(M_{i,j})\}_{j=1,\ldots,N_2}\ 1 \right],\ i = 1, \ldots, M$
       // $N \times 1$ boundary condition vector $B$
19     $B_i = h(i),\ i = 1, \ldots, M$
20     Compute least-squares solution to $AC = B$
21     $n = n + 1$                               // uniformly linearly increasing order
22 end
```

Algorithm 1: The high-level view of the algorithm

3.1.1 Algorithm Details

The first challenge of using this algorithm computationally is the specification of input data in a suitable manner. In this version of the algorithm, I assume the domain to be a potentially non-convex polygon, not allowing for curved boundaries. The boundary conditions are specified in terms of global coordinates rather than parameters like arc-length, which suffices for most problems. In this approximation scheme, it is necessary for the z_j poles to be clustered exponentially according to the theory and the interior point z_* to be away from any poles in the bulk, as shown in Figure 3.1. If z_* is on the outside or near a boundary of the domain, or the poles are not clustered in this fashion, I found it can lead to unexpected numerical instability, as it violates the assumptions of the theory, and doesn't make much practical sense.

I first look in more detail at the selection of poles for the "Newman" part of the approximation, done in lines 3-6. Here, I fix the exponentially clustered poles of the domain. Intuitively, they are clustered in this way to provide higher weighting to the corners. As indicated by the theoretical results of [21] and Theorem 2.3. of [21], they must scale with $\sqrt{n}$ in order to insure root-exponential convergence. According to the results of [21], $\sigma = 4$ was found experimentally to give good speed of convergence. I found this to be accurate when running comparisons, as shown in Figure 3.2. In order to improve convergence, re-entrant corners are given $3n$ poles, vs. non a re-entrant n poles. In order to avoid serious numerical errors, in the case where a domain is non-convex and a pole ends up inside the domain, I just drop it. A better solution could be testing normals which do not intersect the domain, though this is not always possible. I will discuss highly irregular domains further in section 3.2.4

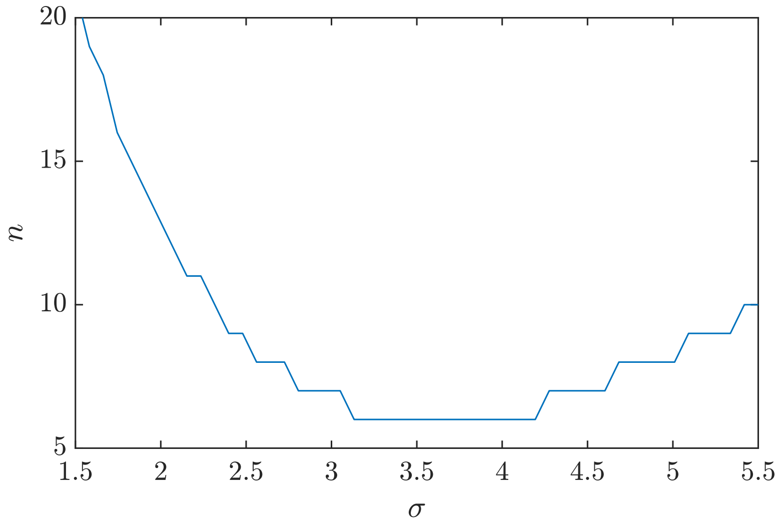

Figure 3.2 Minimal n to reach 10^{-3} error for standard L-domain for $\sigma \in [1.5, 5.5]$

The monomial terms, the "Runge" term of the approximation is relatively simple to evaluate. Assuming the fixed point of expansion within the bulk is correct, I can take the value N_2 to be roughly half of n, a value determined to be sufficient for smooth convergence experimentally, and which exceeds the order needed for theoretical convergence, $\sqrt{n}$. I can then easily derive the value of $N = 2 * N_1 + 2 * N_2 + 1$, which represents the number of real degrees of freedom of the problem.

I find the boundary sampling points based on the positions of the poles. That is, the positioning of the boundary points is determined by the distance of poles from their corners. For each pole, I create 6 boundary samples, placed symmetrically along both sides joining at the associated corner. They are spaced along $1/3,\ 2/3$ and the full distance, where allowable by the side-length. I can see this in Figure 3.1.

In order to actually find the coefficients for this approximation scheme which best describe the boundary conditions, I need to minimise the the supremum norm of the error on the boundary, i.e. $\|AC - B\|_\infty = \sup_{i=1,\ldots,M}(\sum_{j=1}^{N} A_{i,j}C_j - B_i)$, the boundary sampling point with the greatest error. It is possible to provide a better error estimate by an extra, finer boundary mesh after the fact, but I found this was not necessary to allow for convergence in my computations.

Least squares solutions are valid in the case where we are dealing with problems where more data is given than the order of the approximation can compensate for, as discussed in the next chapter. Here I seek to minimise $\|AC - B\|$ to find the best solution to $AC = B$. This is done using the MATLAB "\" operator, which executes an optimised QR-factorisation procedure. Note, other procedures are run for square matrices, but this will seldom be the case for these problems.

QR-decomposition is a standard procedure in numerical linear algebra, so I won't explore this in exhaustive detail. Given an $m \times n$ matrix A, I can decompose it into a unitary $m \times m$ factor, U, and an upper triangular $m \times n$ factor, R.

$$A = QR = \begin{bmatrix} Q_1, Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1$$

where R_1 is $n \times n$ upper-triangular, Q_1 is $m \times n$ and Q_2 is $m \times (m - n)$. Note a unitary matrix has orthonormal columns. I can use several methods to compute this Q and R, including Gram-Schmidt methods, Householder reflections and Givens rotations. MATLAB uses a variant of the Gram-Schmidt method, where the decomposition is multiplied by a permutation matrix, causing the diagonal entries to be non-decreasing, leading to increased stability, as detailed in [14]. Once computed, it can be used to reduce the least squares problem by transforming them into the normal equation, $A^\intercal AB = A^\intercal C$, and then expanding.

$$Q^\intercal(AB - C) = \begin{bmatrix} R_1 B - {Q_1}^\intercal C \\ {Q_2}^\intercal C \end{bmatrix} \implies (AB - C)^2 = (R_1 B - {Q_1}^\intercal C)^2 + ({Q_2}^\intercal C)^2$$

Then to minimise, I look for the solution to $R_1 B = {Q_1}^\intercal C$, which is more easily solved as R_1 is upper triangular. In order to improve numerical stability, one further step is taken here. As noted in [21], it is necessary to re-scale the columns of the matrix A to uniform 2-norm before computations to avoid large entries near corners, as it is easier to handle uniform and smaller values in a precise way with QR algorithm. This does not change the answer as it is equivalent to pre-multiplication by a matrix of the norms, which does not change the normal equation.

3.2 Extensions to Algorithm

3.2.1 Discontinuous Boundary Conditions

When I deal with discontinuous boundary conditions, as discussed in section 2.3, I need to adjust the algorithm to incorporate weighting in order for it to converge. The results of this method are shown in the subsequent chapter. I do this by letting the weights for each point on the boundary be the distance they are from the closest corner, i.e. $w_i = |b_i - \omega_i|$. Other than that, the algorithm should work as normal, with no adjustments to the actual fitting necessary. Interestingly, though not explained in the paper, the implementation published by Gopal & Trefethen also multiplies the approximation terms in the matrix A by these weights for the boundary sampling points, presumably as an additional form of scaling. I found this did not make a difference to my solution in any discernible way.

3.2.2 Curved Boundaries

As discussed in section 2.5, I can handle curved boundaries that do not create particularly non-convex regions without changes to the theory. From an implementation standpoint, these are handled by changing the specification for the boundaries from a straight line to a parameterized curve, with parameter bounds specified. The vertex normals are computed in the same way as normal in my implementation, but to handle more complex boundaries you could take into account the behaviour of the two neighbouring sides, as noted previously where I suggest altering the line of poles to follow a path other than the bisector, still clustered exponentially.

To find the boundary sampling points in this case precisely, it is necessary to either use an arc-length parameterization for the boundaries or to compute one numerically, as this allows us to preserve accuracy in the distance used for placing the points. In my implementation, I assume arc-length is used and handle distance without any major changes consequently.

3.2.3 Neumann Boundary Conditions

As noted in section 2.5, the main alteration necessary to the algorithm is the addition of derivative estimation on the boundary of the domain. I do this using a one-sided derivative estimate, which is sufficient to achieve similar error to the normal upper-limit of accuracy in most cases given in section 4.1. If higher tolerance were desired, it is likely using a more sophisticated algorithm for derivative estimation would be advisable, such as a central derivative estimate, or those discussed in [26].

In one-sided derivative estimation, I simply sample the boundary as normal with a duplicate set of points positioned some small distance, ϵ, along the normal at that point. In my implementation, I let $\epsilon =$ the error tolerance which has shown sufficient accuracy experimentally. This point should be taken within the domain. I then pre-multiply the approximation matrix by a factor, R, specified below row by row, which gives the difference between the solution at these boundary pairs multiplied by the distance, and match this to the boundary condition, i.e. $RAC = B$.

$$R_i = \begin{cases} \epsilon \quad \begin{pmatrix} 0 & \dots & 0 & 1 & -1 & 0 & \dots & 0 \end{pmatrix}, & \text{if } b_i \text{ is a Neumann boundary condition,} \\ \quad\;\; \begin{pmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{pmatrix}, & \text{if } b_i \text{ is a Dirichlet boundary condition.} \end{cases}$$

A central derivative estimate method would involve another additional sample point symmetrically positioned outside of the domain. I have not tested this method, but suspect it could lead to issues if a pole is nearby, i.e. as you near a corner.

3.2.4 Other Potential Extensions

Here I discuss potential approaches to a series of other improvements which are interesting, which either I have determined via consideration of the theory from before or have been suggested by the literature. They all present some new computational challenge or aspect to the basic algorithm presented at the beginning of this chapter. They are mostly unimplemented, and in various states of consideration.

Adaptive Order of Approximation

In algorithm 1 line 21, I see that order of approximation is increased linearly for all poles until convergence is reached. With something like the following stub, it is possible to have an adaptive approach. I did not use this in my implementation as though interesting, it only increases speed of operation fractionally in most cases. It would be quite effective in the case of a larger domains where extra points are added to handle detail however, allowing the approximation to focus on the problematic areas.

```
  // Same as for algorithm 1 until end of loop
1 $n = 3$                     // Constant determining order of approximation
2 Compute mean error per corner, weighted by distance to boundary sample points.
3 **if** mean error for corner tolerance $> \epsilon$ **then** increase order of approximation at this
    corner
4 **else** maintain current level of accuracy
  // Same as for algorithm 1 after end of loop
```

Algorithm 2: Adaptive Accuracy Algorithm

Alternative schemes include checking if maximal error on neighbouring edges is below tolerance, if increasing the order of the corner decreases mean error weighted by distance, or increasing the order of the approximation for some top percentile of weighted error for corners.

Chapter 4

Performance

In this chapter, I will discuss and analyse the performance of the algorithms discussed in chapter 3. I will demonstrate the results of the algorithms on standard domains, discuss convergence, accuracy of the generated results, stability of the methods, and make comparisons to the current standard methods for this kind of PDE simulation. All these simulations were able run quickly using MATLAB on a consumer laptop. What may be feasible or acceptable may change for large-scale computations on a cluster. In particular, the machine precision limits of the algorithm could likely be addressed using arbitrary precision arithmetic methods, which were too slow to run on my devices.

In Figure 4.1, you can see the standard output of my program. Note that my graphs are based on those of [21] in order to allow for easy cross-comparison of my implementation and modifications to their implementation of the algorithm. To summarise, on the right you can see the estimated solution to the boundary value problem provided. Here I show a simple Dirichlet problem, with analytic solution $\Re\{\exp(z)\}$, and the resulting boundary values. Other examples are discussed later to demonstrate specific phenomena and other standard examples are shown in Appendix B.

The top left figure shows the error along the boundary of the domain, relative to the angle made with the center point, shown as a black dot within the bulk on the right. In this case, you can see the error decreases as the corners are approached, and is higher in areas of high boundary detail on the interior of an edge. Intuitively, this is due to there being fewer points of interpolation. By the maximum principle for harmonic functions, this error on the boundary is maximal within the domain, but it is still interesting to see the error within the domain when we know the analytic solution, which I will show and discuss in section 4.1.

Finally, in the bottom left you can see the convergence curve for this problem, with log-error plotted against the square root of the degrees of freedom in our approximation, a.k.a. the approximation order. You can see in this case there is a clean straight line graph indicating root exponential convergence in line with the theory of section 2.1. For other domains and boundary value problems, this convergence can be slower, with gentler slope, but the relationship is maintained for the basic problem in every case tested.

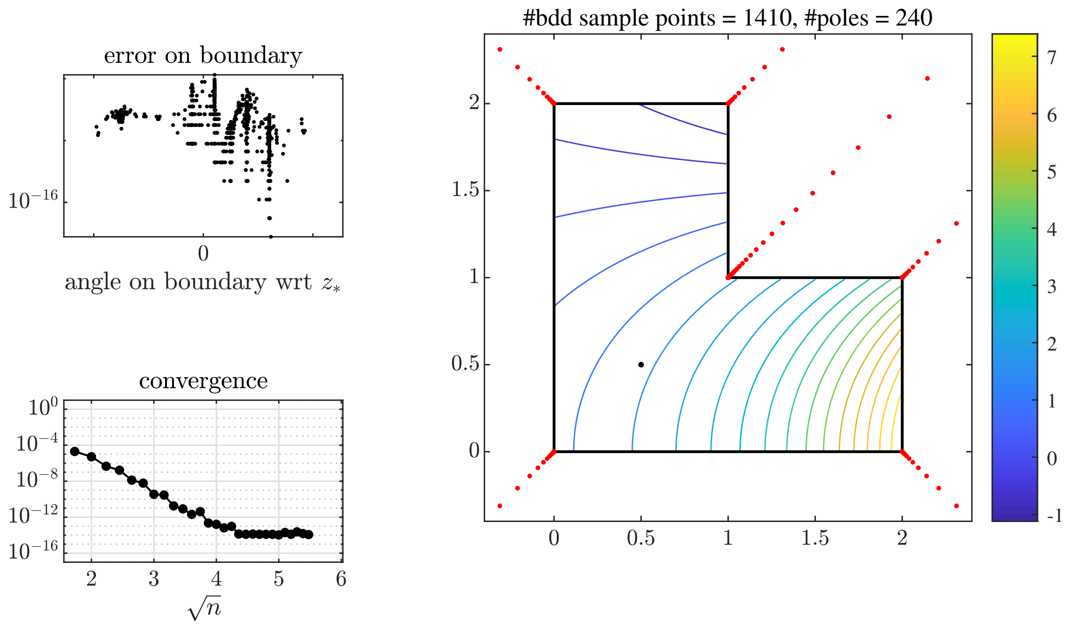

Figure 4.1 Result of program for $\Re(\exp(z))$

The other features discussed in chapter 3 are shown in Figure 4.2. First shown is the discontinuous case, where the solution jumps between edges as having a value of 1 or 0 respectively. Here, I weight error by distance to the discontinuity as discussed. Otherwise metrics are the same, and convergence is similar, if slower and with a lower cap on accuracy discussed shortly.

The next example shown is the Neumann boundary case, specifying derivatives at the boundary such that they are continuously consistent with the Dirichlet conditions also specified. Note, without further modification, it isn't possible to use this method for purely Neumann boundary conditions, as the solution is not uniquely defined. It is possible to arbitrarily specify the value at some interior or boundary point in this case, which will solve uniquely and can be varied to provide all the solutions to the problem in the family. In this case, convergence is not root-exponential. While I was not able to determine why, I suspect this is due to the method I use for derivative estimation (a single-sided estimator), introducing additional instability. For a more thorough treatment, it is perhaps necessary to use a methods such as in [26].

The curved boundary case behaves similarly to normal, with slower convergence. Note that it is necessary to take extra care in this case, adding an additional "corner" on the long side to provide the necessary precision. I present the convergence plots with and without this additional corner in Figure 4.3.

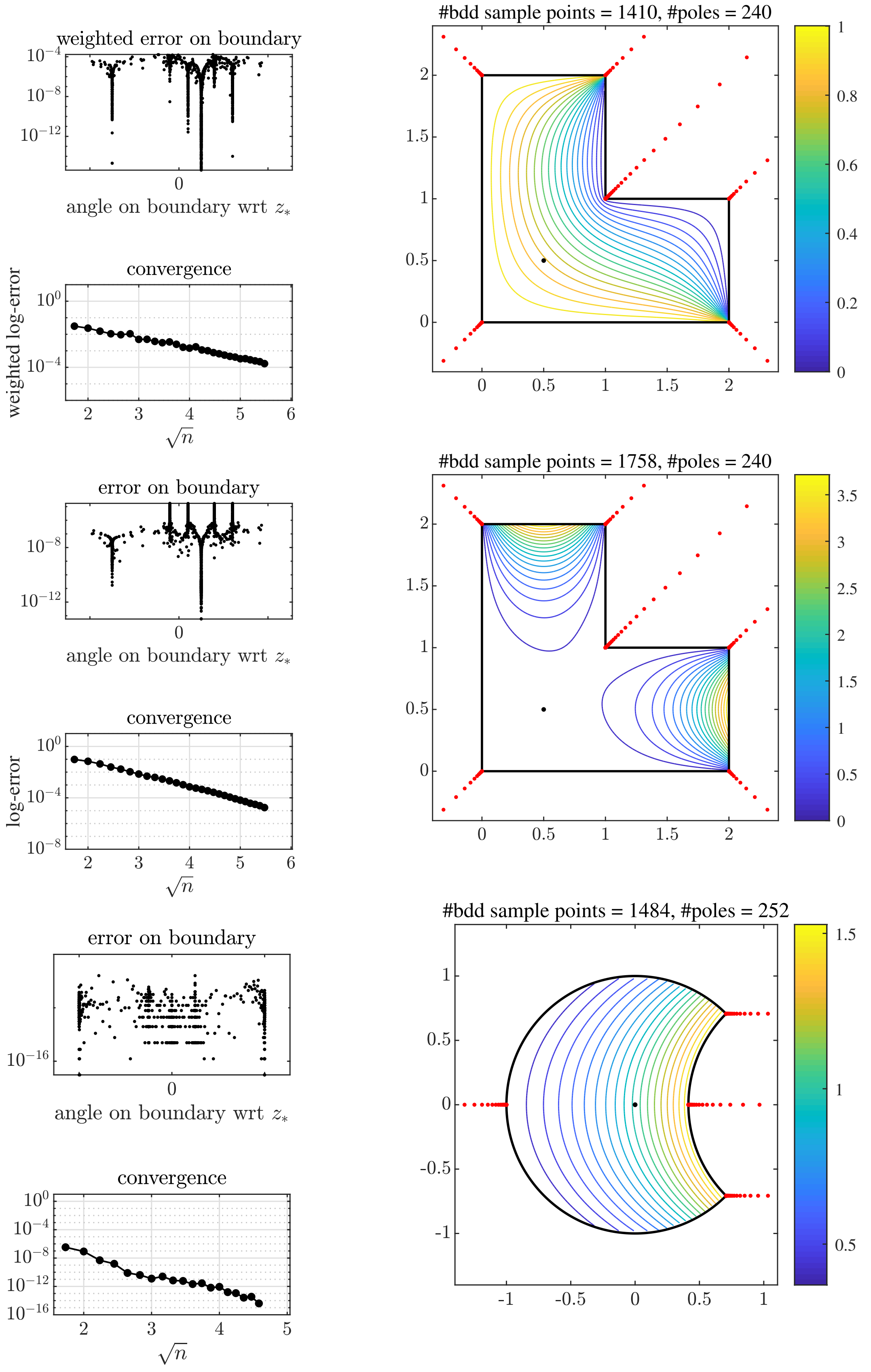

Figure 4.2 Top to bottom, discontinuous with bdds of 0 and 1, Neumann with bdd 1 on top/right, and curved boundary for $\Re(\exp(z))$

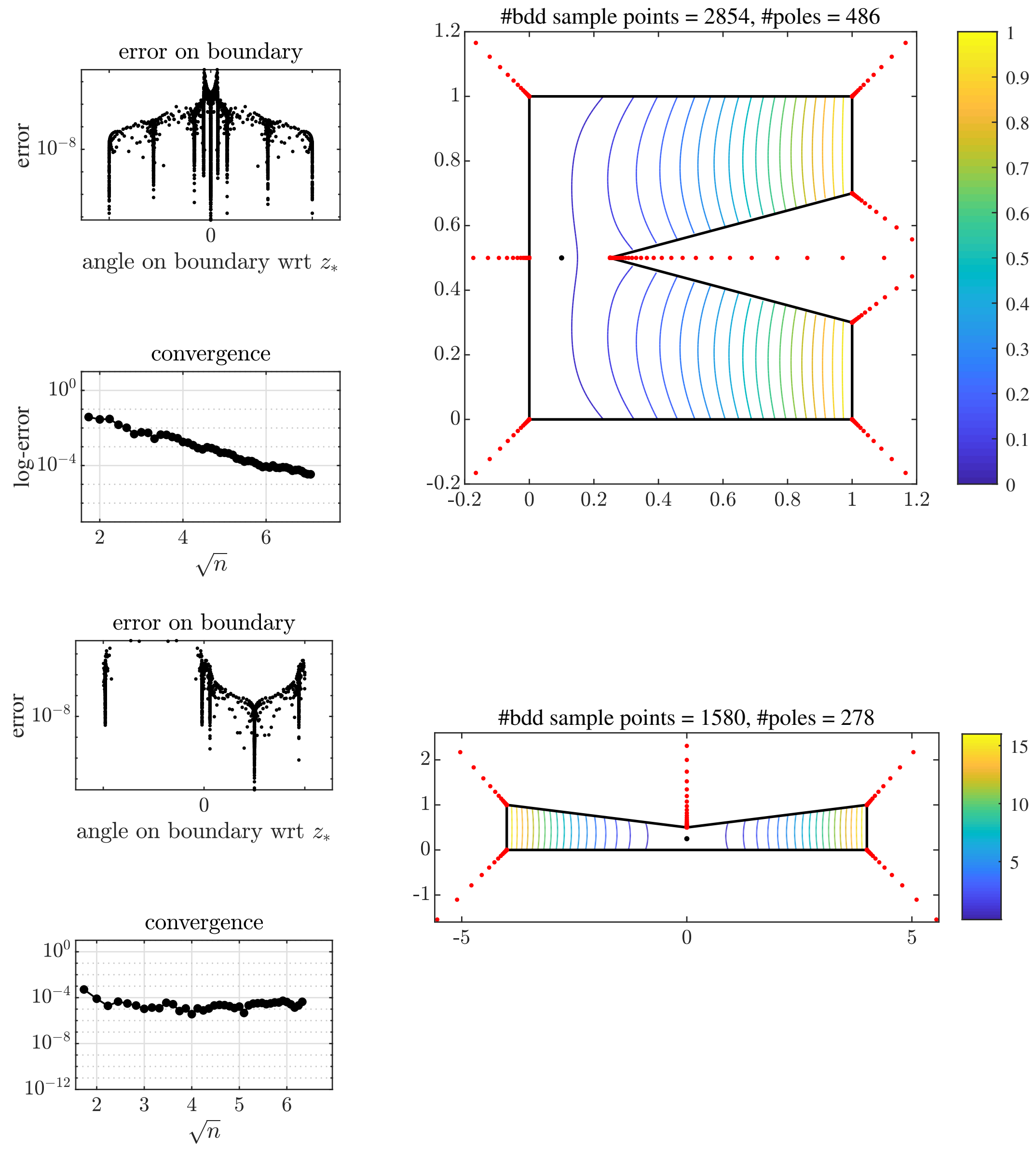

Figure 4.3 Problem Cases for boundaries of $\Re(x)^2$ on re-entrant spike (top) and elongated domain under same conditions(bottom)

Finally, there are two main cases where the solution begins to break down. Those are highly convex domains, like the re-entrant spike, and highly elongated domains, such as the re-entrant spike stretched horizontally. I show these examples in Figure 4.3. The re-entrant case can be attributed to the behaviour within the domain being less smooth - the curvature in the region of the spike becomes more dramatic as it grows closer to other edges within the domain, and as it approaches a point. This is shown mainly by an early stagnation of the convergence - it is still root-exponential but much slower than usual, with an accuracy cap of 10^{-5}. I discussed potential resolutions to this in the previous chapter, but it is not a major problem.

In the elongated case, intuitively you can see that the exponential clustering becomes an issue as the distance between boundary sampling points becomes greater. It seems to cause a near immediate stagnation in maximal accuracy, though with still workable results at about 10^{-4}. A potential solution here is adaptive "corner" insertion (placing poles along the edge), which I test later after considering general accuracy caps.

4.1 Accuracy

While the maximum principle guarantees the error on the boundary bounds the error within the domain of the approximation, it is of interest to consider the behaviour of the error within the domain, when this can be evaluated. For a problem where I take a boundary value problem specified by a specific closed form holomorphic function, I can plot a heatmap of the error within the domain, allowing us to verify the maximum principle as well as understand the behaviour of the error within the bulk.

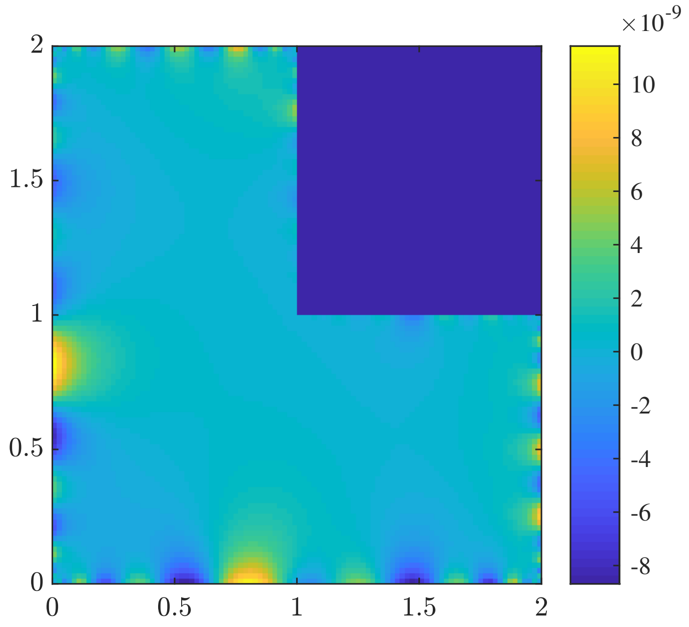

Figure 4.4 Error Heatmap for $\Re(\exp(z))$, where max. error on boundary is 6×10^{-9}, $n = 8$. You can see the effect of poles, and the need for a slightly finer sampling mesh to get an exact error estimate on the domain.(Ignore the square in top right, the region is undefined)

Here we can see the slight issue with not taking a finer boundary sampling mesh for post-processing error checking, as we have areas in the domain between the sampling points which are slightly greater than the maximal error predicted. This can be solved by adding sampling points half-way between the existing ones, as we see these are the areas of maximal error on the domain. This does not violate the maximal principle here, as we see the maximal error is still on the boundary, with a trend towards accuracy within the bulk of the domain.

While the solution converges in all cases but the Neumann case with root-exponential convergence, it does reach an implicit limit in accuracy for all problems at machine precision, which is the highest accuracy representably by a 32-bit floating point number on a computer. This is in the range of 10^{-16}, and can be seen in more straightforward examples such as the real part of the exponential function on most domains. At this point, I cannot gain further precision without using arbitrary precision arithmetic, which is comparatively very slow in MATLAB. A reimplementation in a dedicated arbitrary precision linear algebra setting, such as mpmath or ALGLIB, could allow for this, but is likely unnecessary for most practical cases. This effect is expected, but it can be seen in Figure 4.5 that lower accuracy caps are present for more complicated cases which do not have easily findable analytic solutions.

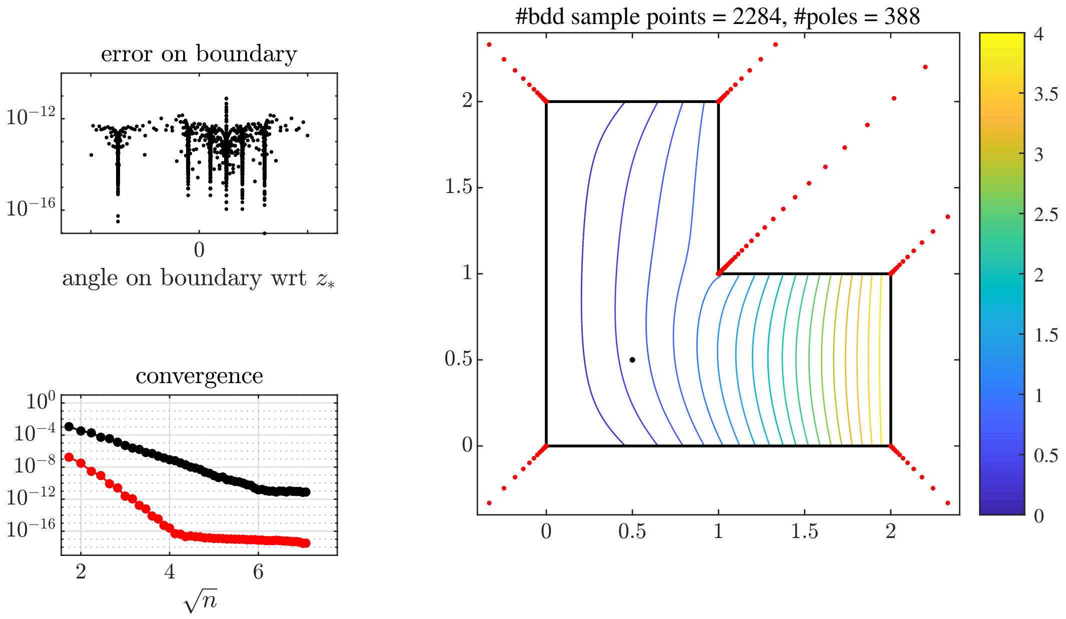

Figure 4.5 Precision limit for boundary values of $\Re(z)^2$, inverse condition numbers are shown in bottom left in red, which reach machine precision prior to accuracy cap.

For the case shown, which is a non-closed form solution version of the L-shaped domain there is an upper limit to accuracy of 7.1418×10^{-12}. This is likely due to internal factors of the calculation exceeding machine precision. Despite the QR factorisation approach, there are still issues due the matrix being “ill-conditioned”. This is mentioned by [21], and a diagnostic suggested to explain this is the condition number of the matrices involved. The condition number of a problem is a measure of the sensitivity of output to small change in input. A higher condition number indicates more sensitivity and the problem is then referred to as “ill-conditioned”. If a problem is well-conditioned, you can expect a backwards stable algorithm to handle it well. This number can be used to estimate the error generated by propagation of rounding errors and numerical input inaccuracy. In statistics, it is associated with co-linearity in regression problems.

For a linear inverse problem with existing inverse, $Ax = b$, the condition number is defined to be the maximum ratio of relative error in x to relative error in b, i.e. for e the error in b, the ratio is

$$\frac{\|A^{-1}e\|}{\|A^{-1}b\|} \Big/ \frac{\|e\|}{\|b\|} = \frac{\|A^{-1}e\|}{\|e\|} \frac{\|b\|}{\|A^{-1}b\|}$$

Its maximum value will simply be $\|A^{-1}\| \, \|A\|$, where I use the operator norm to absorb the maximisation, depending on the norm of the space. In this application, we take the effective condition number, which is the ratio of the largest to smallest non-zero singular values of the matrix A. In Figure 4.5, you can see the condition number reaches machine precision shortly before the problem ceases to converge. However, more ill-behaved problems there is indeed convergence beyond this point.

As noted in [21], the fact it continues to converge could potentially be explained by the effect of over-complete bases, as discussed in [22]. Here, the concept of frames are introduced, which are spanning sets (the term complete is used for spanning sets in functional analysis) satisfying the *frame property*, where our underlying space is some function space S, our frame is $\{\phi_1, \dots\}$, and I fix A, B s.t. $0 < A \leq B < \infty$.

$$A \left\| f \right\|^2 \leq \sum^{i} n f_{k=1} \langle f, \phi_k \rangle^2 \leq B \left\| f \right\|^2, \forall f \in S$$

This property notably shows that any function in our space can be represented as a linear combination of our frame with bounded coefficients. This is useful as if dealing with an ill-conditioned problem, which is near-singular, then finding a unique solution is impossible. When you relax the condition and use an over-complete (non-linearly independent) frame, you can find many near-optimal solutions by making the problem less stringent.

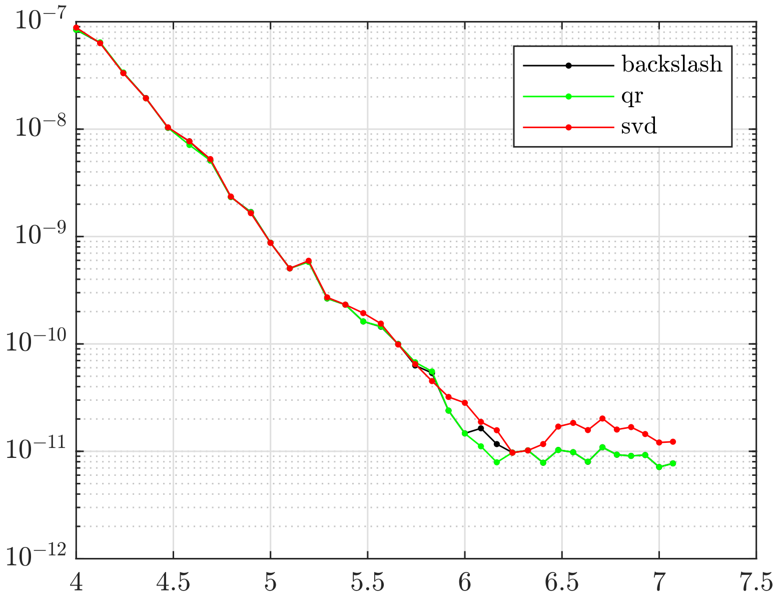

Figure 4.6 Precision limit for boundary values of $\Re(z)^2$, for MATLAB backslash, Truncated QR factorisation, and truncated SVD.

You may have noticed several other convergence lines plotted above. In [22], it is noted that MATLAB's backslash operator does not necessarily follow the theory of the paper, and thus they use truncated variants of direct QR and singular value decomposition methods. In these methods, I drop the smallest singular values, or columns of the QR decomposition, setting them to zero and considering only some truncated percentile - this allows for a more well-posed problem without losing precision, sometimes leading to a net gain in accuracy. The percentile truncated requires further experimentation, but you can see the slight improvement. It is likely that since the publication of the paper, MATLAB has introduced truncation into their backslash method, as it is identical to the truncated QR method. Ultimately the problem requires transformation or re-framing to handle the particularly poor cases such as L-shaped domain with complicated boundaries, but you can see some extra precision using these methods, perhaps greater gains if approached optimally and rigorously.

In the issue above, with an elongated domain, you see the accuracy of the solution dropping off earlier than expected. With adaptive corner creation, I can mitigate the problem. As mentioned previously, by this I mean the addition of corners on the edges of the domain where there is a straight line, in order to add additional sampling points on the boundary. Other methods are possible, but this should allow for an idea of whether the addition of further boundary sampling points is useful. In Figure 4.7, you can see that the decrease in maximal precision relative to diameter is linear, though steep, and that there is nearly no decrease when the adaptive methods are applied, under the supremum norm as usual.

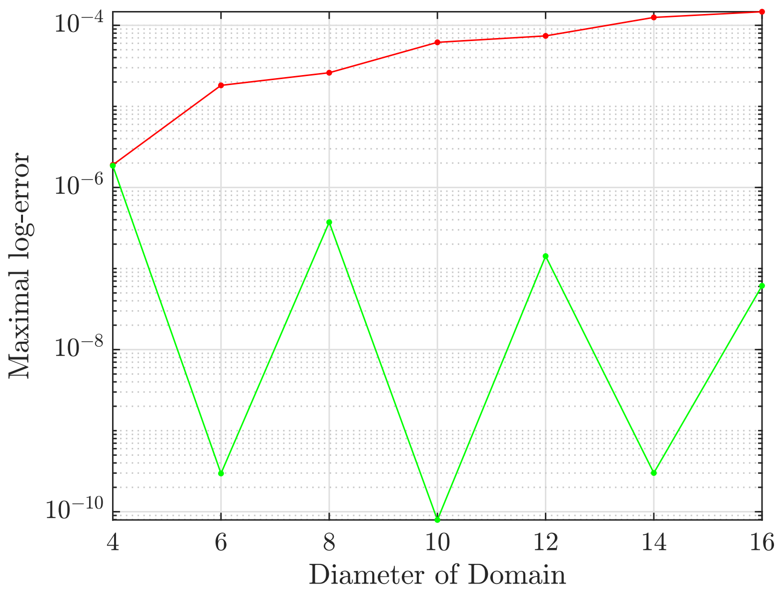

Figure 4.7 Adaptive Precision Methods for Elongated Domains

4.2 Comparison to Standard Methods

As a point of comparison, I present the performance of the two main competing approximation methods here - the boundary element and finite element methods. Code for these reference implementations is found in Appendix A. The boundary element method is based on an integral equation formulation of the problem, and as such is only applicable if you can calculate Green's functions (the fundamental solutions). If this is possible, as it is for the Laplace problem, you get the fundamental solution of

$$u^*(x,y) = \frac{1}{4\pi} \ln\left((x-x_0)^2 + (y-y_0)^2\right)$$

As explained in [16], you can derive from this using Green's identities the boundary integral equation form of the problem, that

$$\lambda u(x_0,y_0) = \int_{\partial} \Omega \left(u\frac{\partial u^*}{\partial \hat{n}} - u^* \frac{\partial u}{\partial \hat{n}} \right) ds$$

where $\lambda = 1/2$ on the boundary, 1 in Ω and 0 elsewhere, and $\hat{n}$ is the unit normal vector to the boundary at the point s. If you evaluate the integral at the boundary you can discretely approximate the integral and thus compute values within the domain. This involves numerical approximation to integrals, which is done via quadrature methods in [16]. I derive my reference implementation from this paper, modifying it slightly to handle my boundary conditions and vectorize(parallel process) it further. Issues with this method include a lack of resolution as you approach the singularities of the problem, but they handle the bulk of the problem well. Also, the method is very slow when evaluating the entire domain, as each point evaluated requires quadrature approximation of the integral equation.

The finite element method involves discretising the domain into a mesh of smaller, locally influencing regions, then solving an approximate local version of the problem on these regions. You can see how the error from these solutions is mainly found towards areas of high detail at the corners. The issue with this method is the approximation of areas with more detail requires a heterogeneous mesh, in terms of cell-size. Here, I do this using the MATLAB PDE toolbox FEM implementation, but many other more sophisticated frameworks for finite element analysis exist.

To compare my findings here with those of Gopal and Trefethen's industry survey, they show that these general methods are quite competitive in most relevant areas and are more extensible, but are typically much slower and require further specialisation for optimal performance on 2-D polygonal domains with potential singularities in boundaries. This was similar to the their findings, though apparently with a high degree of additional refinement such problems can be approached well with the general methods.

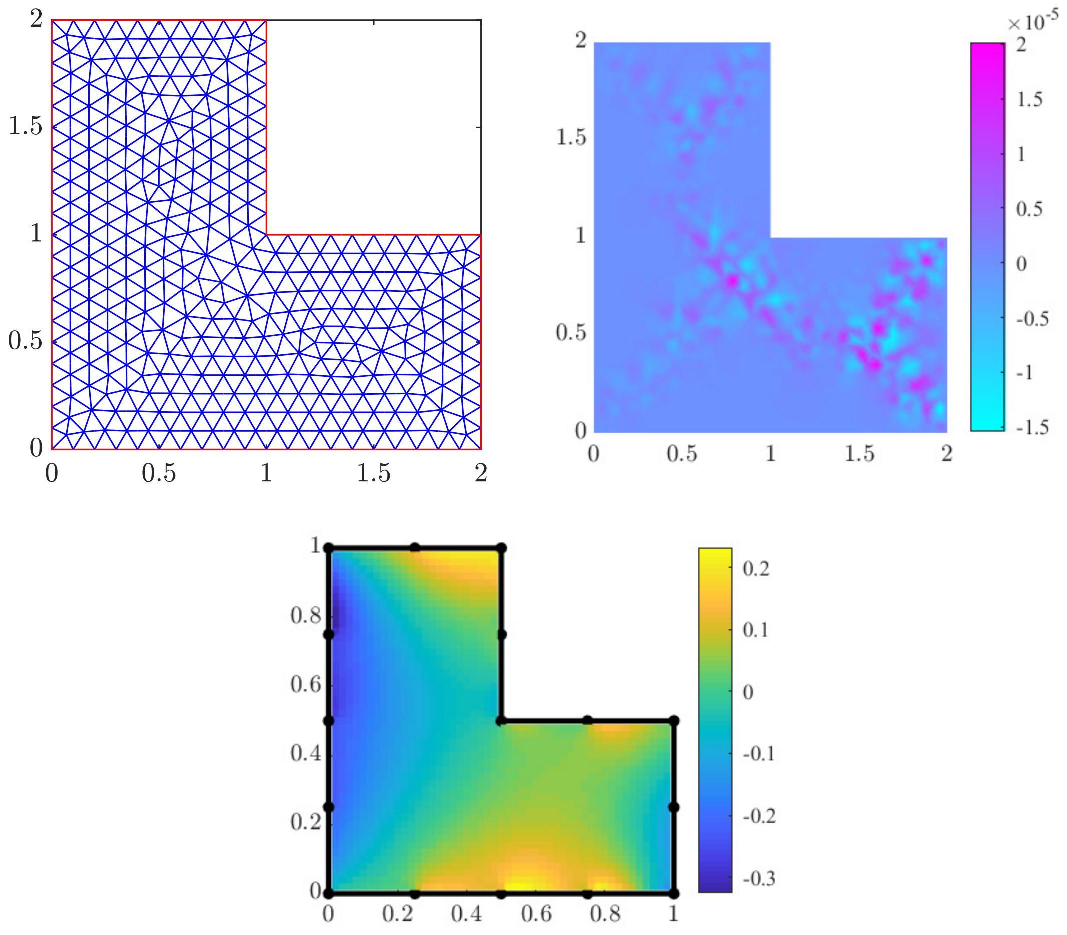

Figure 4.8 FEM mesh (top-left), and error map (top-right), BEM error map (bottom)

Chapter 5

Concluding Remarks

I have now discussed the theoretical and practical effectiveness of the method proposed by Trefethen, as well as several extensions. Several more applications of the theory are addressed here, as they would provide significant improvements to the applicability of the method.

Slits

The issue, proposed by Gopal & Trefethen, is that for a domain which is a disk with a single slit removed you cannot place poles along said slit. They suggest the use of a local transformation, taking $\zeta = \sqrt{z}$ for a slit along the negative axis, taking it to the positive imaginary axis. You then could place the poles along the positive imaginary axis and solve the least squares problem with modified scheme, $1/(\sqrt{z} - \zeta_k)$, where ζ_k are poles. As we can always perform a local transformation taking a line through the origin to the positive imaginary axis, and this handles the case of a slit for the local result, this observation should apply to all problems with slits.

Transmission Problems

The implementation of problems where the rational approximations on two different domains are linked via a point, where they must share derivatives or values. Likely this would require two different approximations, subject to some additional constraint due to the transmission during the least squares optimisation step.

Multiply Connected Domains

For a domain with holes, poles need to be placed within said hole. As considered in the case of highly non-convex shapes, this is a challenging problem. Additionally, it will be necessary to adjust the theory to use the more general form of the Runge approximation theorem, where the approximations will now be entirely rational.

Three Dimensions

Three dimensional problems present major issues for this method, as the majority of the complex analytic results used do not hold in general three-space, and representations for many concepts must change. To begin with, the solutions to the Laplace equation are still harmonic functions, but of greater dimension, and thus have fewer nice properties. However, the maximum principle still holds, and as such we need only consider the boundary for the purpose of the error estimation. It is likely that inspiration could be drawn for an extension here from the method of fundamental solutions in three dimensions, similar to the boundary element method.

Further Recommendations

I also have several looser recommendations, noting things I missed or had insufficient time/resources for in this project. Many have been mentioned previously, but I bring them together here. Firstly, a more sophisticated treatment of the Neumann boundary problem would likely yield results of higher quality. I would be very interested to see a better single-sided derivative expansion applied in the way I proposed earlier, such as discussed by [26].

Next, it is possible that though this method is the optimal rational approximation under the minimax error, it would be possible to yield better results using a hybrid approach with another method, such as BEM, FEM or the method of fundamental solutions. In the areas approaching corners where these methods suffer, this method could be used as a faster and more accurate estimate, but getting the benefits of a full discretisation such as fewer error peaks on the boundaries. Perhaps even a different sampling approach away from the corners (non-exponential)could resolve these clusters of error on the boundary.

In this vein, the theory of potentials should really be fully explored for the case of non-convex domains. It seems incredibly promising, I am disappointed that I lacked the full machinery to approach it. Better sampling yielded by this could easily help handle slowdowns in convergence and early performance caps seen for the re-entrant spike.

Furthermore, I would be very curious to fully determine the cause of these performance caps, and run sufficient diagnostics/profiling within MATLAB to verify where it runs into machine precision, or confirm this is the case with a more optimised arbitrary precision linear algebra library. A more high-powered computer doing an analysis with this and my FEM/BEM simulations would yield more accurate results and useful diagnostics, as my machine was not able to simulate it for sufficiently high degrees of detail to generate useful convergence curves without heavy optimisations I did not have the expertise for.

Finally, I would like to thank the reader for taking the time to explore this fascinating subject with me.

Appendix A

Annotated Matlab Code

You can also find runnable versions of this code at https://github.com/caelen-feller/fyp. I have omitted some excessively verbose MATLAB plotting code here, as there are many constants necessary to set in order to export figures properly to LaTeX.

Laplace Solver Standalone Code

This is my implementation of the algorithm described by Gopal & Trefethen, and is described in algorithm 1 at a higher level. For truncated QR-solving, I use a RRQR implementation from the MATLAB file exchange by Xin Xing for demonstration purposes.

```
% Laplace equation solver on polygonal domains
% Arguments:
%   w = List of boundary vertices, counterclockwise
%   center = Point centrally situated within domain used for
   interior
%   polynomial-based approximation
%   h = Boundary value problem (Dirichlet), specified per-edge
   as functions
%   of the points on the boundary
%   tol = Desired accuracy of solution
% Return Value:
%   u = Solution to boundary value problem, valid on the domain
     only
% Optional Named Parameter Pairs:
%   'tests' = false, Turn on visual inspection of boundary and
   poles before
%   computations.
%   'neumann' = false, Turn on support for neumann boundary
   conditions.
```

```matlab
%   Note, this does not work with solely neumann, at least one
   must be
%   dirichlet. Specify a side is neumann by adding 1 to second
   column, 0 if
%   dirichlet, i.e. [{@(z) z} 1; repmat([{@(z) 0} 0]), length(w
   ), 1]]
%   'discont' = false, Weights approximation and error by
   distance to
%   nearest corner, allowing for convergence with corner
   discontinuities.
%   'curved' = false, Allows for curved boundaries, specified
   in curves.
%   Does not support neumann boundaries at same time currently.
%   'curves' = [], Parameterisation of curved boundaries.
   Example row would
%   contain [{@(t) exp(1i*(t+pi/4))} 1.5*pi 1], the function,
   maximal value
%   for t and the normal at the corner.
%   'curved_hull' = [], A polygonal representation of the
   boundary for use
%   in inclusion testing and plotting.
%   'plot3' = false, Plot using contours or 3D representation.

function u = laplace_solver(w, center, h, tol, varargin)
    p = inputParser; % Handle optional parameters
    addOptional(p, 'tests', false);
    addOptional(p, 'neumann', false); % Does not support curves
        yet
    addOptional(p, 'discont', false);
    addOptional(p, 'curved', false)
    addOptional(p, 'curves', []);
    addOptional(p, 'curved_hull', []);
    addOptional(p, 'plot3', false);
    addOptional(p, 'plots', true);
    addOptional(p, 'fixed', false);


    parse(p, varargin{:});
    neumann = p.Results.neumann;
    curved = p.Results.curved;
```

```
if curved, curves = p.Results.curves; end
if curved && neumann, disp("Error: Curved domains are not
   supported with neumann boundaries yet"); end

% Get the maximal diameter of the domain
w_r = sort(real(w)); w_r = w_r([1 end]);
w_i = sort(imag(w)); w_i = w_i([1 end]);
scale = max([diff(w_r),diff(w_i)]);

if curved % Get hull-based scale
    b_r = sort(real(p.Results.curved_hull)); b_r = b_r([1
       end]);
    b_i = sort(imag(p.Results.curved_hull)); b_i = b_i([1
       end]);
    scale = max([diff(b_r),diff(b_i)]);
end

% Confirm validity of poles and boundary sampling points if
    desired
if p.Results.tests
    clf
    poles = compute_poles_test(w, scale, 3, center, p);
    y = input('next? ');
    bdd = bdd_sample_test(w, poles, h, 3, tol, center, p);
    y = input('next? ');
end

% Increase n until convergence to specified tolerance.
max_steps = 30;
prev_err = inf; bdd = []; flat_poles = [];
errs = inf(1,max_steps); Nvec = zeros(1,max_steps);
for n = 3:max_steps
    % Exp. clustered poles at corners
    poles = compute_poles(w, scale, n, p);
    flat_poles = cell2mat(poles).';
    [~, N1] = size(flat_poles);

    % Distance of poles from corners and boundary sample
       points
    [bdd, b, dist] = bdd_sample(w, poles, h, n, tol, p);
```

```
[M,~] = size(bdd);
[Ms, ~] = size(b);
newman = dist./(bdd - flat_poles);

% Polynomial approx in bulk
N2 = ceil(n/2);
runge = ((bdd - center)./scale).^(1:N2);

% Construct least squares matrix
N = 2*N1 + 2*N2;
disp("n = "+n); disp("N = "+N); disp("N1 = "+N1); disp
   ("N2 = "+N2);
disp("M = "+M); disp("Ms = "+Ms);

A = [real(runge) imag(runge) real(newman) imag(newman)
   ones(M,1)];

% If necessary, weight by distance to vertex
weights = ones(1,M);
if p.Results.discont, weights = min(abs(bdd.' - w));
   end

% Handle Neumann Boundaries
D = eye(M, M);
b_neumann = zeros(M,1);
if neumann
    D = zeros(Ms, M);
    zero_i = 1;
    for i = 1:Ms
        if b(i,2) == 1
            D(i,:) = [zeros(1, zero_i-1) 1/tol -1/tol
               zeros(1, M-zero_i-1)];
            b_neumann(zero_i+1) = 1;
            zero_i = zero_i + 2;
        else
            D(i,:) = [zeros(1, zero_i-1) 1 zeros(1, M-
               zero_i)];
            zero_i = zero_i + 1;
        end
    end
```

```
end
weights = weights(b_neumann == 0);
weight_m = spdiags(sqrt(weights.'), 0, Ms, Ms);

% Solve using backslash, matricies are always ill-
   conditioned
warn = warning('off','MATLAB:rankDeficientMatrix');

% Truncated QR-Solver
f = 1.04; %Truncation Constants
[Q, R, perm] = sRRQR(D*A, f, 'tol', tol*10^-2);
[~,perminv] = sort(perm);
c = (R\(Q.'*b(:,1)));
c = c(perminv);

% Normal QR-Solver
[Q,R,perm] = qr(D*A);
c = perm*(R\(Q.'*b(:,1)));

% Normal SVD Solver
[U,S,V] = svd(D*A);
c = V*(S\U.'*b(:,1));

% Truncate it
truncSVD = @(U,S,V,p) V(:,1:p)*diag(1./diag(S(1:p,1:p))
   )*U(:,1:p)';
c = truncSVD(U,S,V,round(sqrt(N)))*b(:,1);

% Backslash solver
c = (D * A) \ (b(:,1));

warning(warn.state,'MATLAB:rankDeficientMatrix');
% Check error on boundary
err_v = weight_m*(D*A*c - b(:,1));
err = norm(err_v,inf);
disp("err="+err);
disp("cond="+cond(D*A));
% Exit if tolerance reached or if error begins to
   increase
```

```
    % as convergence is not possible due to numerical
       instability.
    errs(n) = err; Nvec(n) = n;
    if ~p.Results.fixed && err <= tol %|| err > prev_err*10
        break;
    end
    prev_err = err;
end
errs = errs(Nvec > 0);
Nvec = Nvec(Nvec > 0);

% Construct proposed solution
u = @(z) [real(((z-center)./scale).^(1:N2)) ...
        imag(((z-center)./scale).^(1:N2))   ...
        real(dist./(z-flat_poles))         ...
        imag(dist./(z-flat_poles)) ones(size(z))]*c;

% Plot solution, poles and boundary sampling points
% Plotting Config
LW = 'linewidth'; MS = 'markersize'; FS = 'fontsize';
fs = 8; PO = 'position'; FW = 'fontweight'; NO = 'normal';

if curved, w_r = b_r; w_i = b_i; w = p.Results.curved_hull;
    end
ax = [w_r(1:2); w_i(1:2)] + .2*scale*[-1 1 -1 1]';
axwide = [w_r(1:2); w_i(1:2)] + 1.1*scale*[-1 1 -1 1]';

% Create sampling grid
sx = linspace(w_r(1),w_r(2),100); sy = linspace(w_i(1),w_i
   (2),100);
[xx,yy] = meshgrid(sx,sy); zz = xx + 1i*yy;

% Evaluate solution on grid, discard points not in domain
ff = zz; ff(:) = u(zz(:)); ff(~indomain(zz,w,p)) = NaN;

% Get bounds for color scale, plot solution, domain and
   poles/bdd
clf, shg
levels = linspace(min(min(real(ff))),max(max(real(ff))),20)
   ;
```

```
if p.Results.plots % Begin Plots

if p.Results.tests
     contour(sx,sy,ff,levels,LW, 1), colorbar, hold on;
     plot(w([1:end 1]), '-k', LW,1), plot(flat_poles, '.r',
        MS,7), ...
          plot(bdd, '.g', MS, 7);
     plot(real(center),imag(center),'.k',MS,7), hold off;
     y = input('next? ');
end
clf, shg
axes(PO,[.4 .25 .6 .6])
if p.Results.plot3, plot3(xx,yy,ff), colorbar;
else
     contour(sx,sy,ff,levels,LW,.5), colorbar, axis equal,
        hold on;
     plot(w([1:end 1]), '-k', LW,1), plot(flat_poles, '.r',
        MS,5);
     set(gca,FS,fs-1), plot(real(center),imag(center),'.k',
        MS,6), axis(ax)
end
title(['#bdd sample points = ' int2str(M) ...
     ', #poles = ' int2str(N1)],FS,fs,FW,NO), hold off;

% Plot boundary error
axes(PO,[.05 .6 .25 .2])
errmin = .01*tol;
ws = 'error'; if p.Results.discont, ws = 'weighted error';
    end
axis([-pi pi .0001*errmin 1]), grid on

semilogy(angle(bdd(b_neumann == 0) - center), abs(err_v),'.
    k',MS,4), hold off

set(gca,'ytick',10.^(-16:4:0))
set(gca,'xtick',pi*(-1:1),'xticklabel',{'-\pi','0','\pi'})
set(gca,FS,fs-1), xlabel('angle on boundary wrt wc',FS,fs),
     ylabel(ws, FS, fs)
title([ws ' on boundary'],FS,fs,FW,NO)
```

```
    % Plot speed of convergence
    axes(PO,[.05 .2 .25 .2]); ws = 'log-error'; if p.Results.
       discont, ws = 'weighted log-error'; end
    semilogy(sqrt(Nvec),errs,'.-k',LW,0.7,MS,10), grid on, hold
        on
    errmin = .01*tol; axis([0.9*min(sqrt(Nvec)) 1.1*max(sqrt(
       Nvec)) errmin 10])
    set(gca,FS,fs-1), title('convergence',FS,fs,FW,NO)
    xlabel('sqrt(DoF)',FS,fs), ylabel(ws,FS,fs)
    set(gca,'ytick',10.^(-16:4:0))

    end % End Plotting
end

% Utility to check if point is in domain
function in = indomain(z, w, p)
    if p.Results.curved, w = p.Results.curved_hull; end
    in = inpolygon(real(z),imag(z),real(w),imag(w));
end

% Generate exponentially clustered poles along vertex normals
   of domain
function poles = compute_poles(w, scale, n, p)
    [m, ~] = size(w);
    curved = p.Results.curved;
    if curved, curves = p.Results.curves; end

    % Edge normals
    n_e = zeros(m, 1);

    for i = 1:m
        n_e(i) = (w(i) - w(mod(i + m - 2, m)+1))*-1i;
        n_e(i) = n_e(i)/norm(n_e(i));
    end

    % Vertex normals (external angle bisectors)
    n_v = zeros(m, 1);

    for i = 1:m
        n_v(i) = n_e(i) + n_e(mod(i, m)+1);
        n_v(i) = n_v(i)/norm(n_v(i));
    end

    % Pole generation
    poles = cell(m,1);
    sigma = 4;
    re_entrant = real(n_v).*real(n_e*1i)+imag(n_v).*imag(n_e*1i
       );
    if curved
        re_entrant = -ones(m,1);
        n_v = cell2mat(curves(:,3));
    end
    for i = 1:m
        N = n;
        % Check for re-entrancy
        if re_entrant(i) < 0
            N = 3*n;
        end

        dist = exp(-sigma*(sqrt(N) - sqrt(1:N)))';
        pole = w(i) + n_v(i) * scale .* dist;
        % Check for poles in domain
        poles{i} = pole(~indomain(pole,w,p));
    end
end

% Generate boundary sample points and get pole scale (distance
   to corners)
function [bdd,b,dist] = bdd_sample(w, poles, h, n, tol, p)
    [m, ~] = size(w);
    curved = p.Results.curved;
    neumann = p.Results.neumann;

    if curved, curves = p.Results.curves; end

    if neumann, n = n*2; end

    bdd = zeros(3*n*m,1);
    dist = zeros(3*n*m,1);
    b = zeros(3*n*m,2);
```

```
bdd_i=1; dist_i=1; b_i = 1;

for i = 1:m
    [N, ~] = size(poles{i});
    prev = mod(m + i - 2, m)+1; next = mod(i, m)+1;
    neg = w(prev) - w(i); pos = w(next) - w(i);

    % Setup for neumann bcs
    n_norm = imag(neg) - real(neg)*1i; n_norm = n_norm /
       abs(n_norm);
    p_norm = -imag(pos) + real(pos)*1i; p_norm = p_norm /
       abs(p_norm);
    if neumann
        if h{prev,2} == 1, L_n = 0:1; else, L_n = 0; end
        if h{i,2} == 1, L_p = 0:1; else, L_p = 0; end
    else
        L_n = 0; L_p = 0;
    end

    if curved, neg = curves{prev,2}; pos = curves{i,2}; end
    % loop through poles, checking distance
    for j = 1:N
        dist(dist_i) = norm(w(i) - poles{i}(j));
        for k = 1:3
            t = k/3 * dist(dist_i);
            eps = tol;
            if(t < norm(neg))
                for l = L_n
                    bdd(bdd_i) = t * neg/norm(neg) + w(i)
                       ...
                        + l*eps*n_norm;
                    if curved, bdd(bdd_i) = ...
                        curves{prev, 1}(curves{prev, 2} - t
                           ); end
                    if l == 0
                        b(b_i, 1) = h{prev}(bdd(bdd_i));
                        if neumann, b(b_i, 2) = h{prev,2};
                           end
                        b_i = b_i + 1;
                    end
```

```
                        bdd_i = bdd_i + 1;
                    end
                end
                if(t < norm(pos))
                    for l = L_p
                        bdd(bdd_i, 1) = t * pos/norm(pos) + w(i
                           ) ...
                           + l*eps*p_norm;
                        if curved, bdd(bdd_i, 1) = curves{i,
                           1}(t); end
                        if l == 0
                            b(b_i,1) = h{i}(bdd(bdd_i));
                            if neumann, b(b_i,2) = h{i,2}; end
                            b_i = b_i + 1;
                        end
                        bdd_i = bdd_i + 1;
                    end
                end
            end
            dist_i = dist_i+1;
        end
    end
    bdd = bdd(1:(bdd_i-1),:);
    dist = dist(1:(dist_i-1),:).';
    b = b(1:(b_i-1),:);
end

% Tests for the above functions
function out = compute_poles_test(w, scale, n, center, p)
    LW = 'linewidth'; MS = 'markersize';
    plot(w([1:end 1]), '-k', LW,1);
    out = compute_poles(w, scale, n, p);
    flat_poles = cell2mat(out);
    hold on, plot(flat_poles, '.r', MS,7), plot(real(center),
       imag(center),'.k',MS,7), hold off;
end

function out = bdd_sample_test(w, poles, h, n, tol, center, p)
    [out,b,dist] = bdd_sample(w, poles, h, n, tol, p);
    LW = 'linewidth'; MS = 'markersize';
    hold on, plot(out(:,1), '.g', MS,7);
    plot(real(center),imag(center),'.k',MS,7), hold off;
end
```

Laplace Solver Usage Example

Below is a usage example of the solver for the demos given in this report. It is not exhaustive (not every example is included in full), but should allow for easier recreation of results.

```
% Example of use of framework on various domains
% Define example polygonal domains
% Note: not all bcs will work on domains with non-even number
   of corners
w_square = [1-1i; 1+1i; 1i-1; -1-1i];
w_L = [2; 2+1i; 1+1i; 1+2i; 2i; 0];
w_isodrum = ([1+2i; 1+3i; 2i; 1i+1; 2+1i; 2; 3+1i; 3+2i
   ]-(1.5+1.5i))/1.8;
w_long_no_fix = [-4 4 4+1i .5i -4+1i].';
w_long = [linspace(-4,4,10) (4+1i + (.5i - 4-1i).*linspace
   (0,1,9)) (0.5i + (-4+1i - .5i).*linspace(0.1,1,9))].';

mag= 0.75;
w_spike = [0 1 1+ 0.3i (1-mag)*1+.5i 1+ .7i 1+1i 1i 0.5i].';

w_slit = [0 1 1+ mag*0.5i .5+.5i 1+ (2-mag)*.5i 1+1i 1i 0.5i
   ].';

w_rev_spike = [0 1 1+.3i 1.5+.5i 1+.7i 1+1i 1i].';
w_rand = (exp(2i*pi*(1:10)/10).*(.1+rand(1,10))).';

% Give corners and arclength param of edges for curved domains
w_circular = [exp(1.75i*pi); -1; exp(.25i*pi); sqrt(2)-1];
circular = [{@(t) exp(1i*(t+pi/4))} 1.5*pi/2 1; ...
    {@(t) exp(1i*(t+pi/4+1.5*pi/2))} 1.5*pi/2 -1; ...
    {@(t) exp(1i*(5*pi/4 - t)) + sqrt(2)} pi/4 1; ...
    {@(t) exp(1i*(pi - t)) + sqrt(2)} pi/4 1;];

w_tri = [exp(.25i*pi); -1; exp(1.75i*pi); -0.5];
tri = [{@(t) exp(1i*(t+pi/4))} 1.5*pi/2 1; ...
    {@(t) exp(1i*(t+pi/4+1.5*pi/2))} 1.5*pi/2 -1; ...
    {@(t) exp(1.75i*pi)- t*(0.5 + exp(1.75i*pi))} 1 1; ...
    {@(t) -0.5 + t*(exp(.25i*pi) + 0.5)} 1 1];

curved = false; curves = circular;
% curved = true; curves = tri; % example for it on with w_tri
w = w_square;

% Need boundary conditions, specified as real funcs of boundary
    values
% Could also specify as set of functions on arc length

h_linear = repmat({@(z) real(z)}, length(w), 1);
h_quad = repmat({@(z) real(z)^2}, length(w), 1);
h_abs = repmat({@(z) abs(z)}, length(w), 1);
h_sin = repmat({@(z) real(exp(z))}, length(w), 1);
h_blowup = [ {@(z) 1/(imag(z)-1.1)}; {@(z) 1/(real(z)-1.1)}
   ;{@(z)1/(real(z)-1.1)} ;{@(z) 1/(imag(z)-1.1)}];

% These all require weighted=true to converge properly
% h_discont = [repmat({@(z) 1}, length(w)/2, 1);...
%     repmat({@(z) 2}, length(w)/2, 1);];
% % These require neumann support to be on
h_r_simple = [{@(z) 1} 1; repmat([{@(z) 0} 0], length(w) - 1,
   1)];
h_r_demo = [{@(z) 10} 1; {@(z) 0} 0; {@(z) 0} 0; {@(z) 10} 1;
   ...
    {@(z) 0} 0; {@(z) 0} 0;];


h_r_test = [{@(z) 1} 1; {@(z) real(z)} 0; {@(z) -1} 0; {@(z)
   real(z)} 0];
weighted = false; neumann = false;
h = h_quad;

% Creates a uniformly sampled hull of the boundary for curved
   domains
% Used in inclusion testing and plotting
n_slices = 100;
curved_hull = [];
if curved
```

```
    [m, ~] = size(w);
    for i = 1:m
        prev = mod(m + i - 2, m)+1; next = mod(i, m)+1;
        curved_hull = [ curved_hull ...
            curves{i,1}(linspace(0,curves{i,2},n_slices))];be
    end
    curved_hull = curved_hull.';
end

% Center of domain (used as sample point for the polynomial
   terms)
w_c = 0;%0.1 + 0.5i;

tol = 1e-3 % Desired accuracy of solution

u = laplace_solver(w, w_c, h, tol...
    ,'tests', true ...
    ,'neumann', neumann ...
    ,'discont', weighted ...
    ,'curved', curved ...
    ,'curves', curves ...
    ,'curved_hull', curved_hull);
```

Other Plotting Code

BEM & FEM Methods

BEM

Note, this is based heavily on the code of [16].

```
% Sample implementation of Boundary Element Method / Boundary
   Integral
% Equation Method on an L-Shaped domain, using LS methods
% based on paper and code by Ang(2008) by Caelen Feller, 2020
clear

% Problem Setup
fileid = fopen('data/bem_input.dat', 'r');
indata = fscanf(fileid, '%g%g%d%g', [4, inf]).';
```

```
% w_L = [0; 2; 2+1i; 1+1i; 1+2i; 2i];
% l = 2;
% w_refined = zeros(length(w_L).*l,1);
% for i = 0:length(w_L)-2
%     for j = 0:l-1
%         w_refined(i*l+j+1) = w_L(i+1) + j*1/l*(w_L(i+2) - w_L
   (i+1));
%     end
% end
% i = (length(w_L)-1);
% for j = 0:l-1
%         w_refined(i*l+j+1) = w_L(i+1) + j*1/l*(w_L(1) - w_L(i
   +1));
% end
% w_refined = w_refined([1:end end]);
xb = indata(:,1).*2;
yb = indata(:,2).*2;
bt = zeros(size(xb));
bv = real(exp(xb+1i.*yb));%indata(:,4);
n = length(xb) - 1;

% Finding midpoints and lengths of elements and unit normal
   vectors,
% assuming only vertical and horizontal line segments
xm = 0.5*(xb(1:n) + xb(2:n+1));
ym = 0.5*(yb(1:n) + yb(2:n+1));
lm = sqrt((xb(2:n+1) - xb(1:n)).^2 + (yb(2:n+1) - yb(1:n)).^2);
nx = (yb(2:n+1) - yb(1:n))./lm;
ny = (xb(1:n) - xb(2:n+1))./lm;

% Processing
% Utility functions
intf = @(t, xi, eta, xk, yk, nkx, nky, lk) ...
    log((xk-t*lk*nky-xi).^2 + (yk + t*lk*nkx-eta).^2);
intg = @(t, xi, eta, xk, yk, nkx, nky, lk) ...
    (nkx*(xk-t*lk*nky-xi) + nky*(yk + t*lk*nkx-eta))...
    ./((xk-t*lk*nky-xi).^2 + (yk + t*lk*nkx-eta).^2);

findf = @(xi, eta, xk, yk, nkx, nky, lk) (lk/(4.0*pi))*...
```

```
    integral(@(t) intf (t, xi, eta, xk, yk, nkx, nky, lk),0, 1)
       ;
findg = @(xi, eta, xk, yk, nkx, nky, lk) (lk/(2.0*pi))*...
    integral(@(t)intg (t, xi, eta, xk, yk, nkx, nky, lk),0, 1);

for m = 1:n
    b(m) = 0;
    for k = 1:n
        if(k == m)
            G=0; del = 1;
            F = lm(k)/(2*pi)*(log(lm(k)/2)-1);
        else
            F = findf(xm(m), ym(m), xb(k), yb(k), nx(k), ny(k),
                lm(k));
            G = findg(xm(m), ym(m), xb(k), yb(k), nx(k), ny(k),
                lm(k));
            del = 0;
        end
        if(bt(k) == 0)
            A(m,k) = -F;
            b(m) = b(m) + bv(k)*(-G+0.5*del);
        else
            A(m,k) = G - 0.5*del;
            b(m) = b(m) + bv(k)*F;
        end
    end
end

z = A\b.';

% Approximate bdd values
u = (1-bt(1:n)).*bv(1:n) + bt(1:n).*z(1:n);
q = (1-bt(1:n)).*z(1:n) + bt(1:n).*bv(1:n);
err = max(abs(u-bv(1:n)));

% Post-Processing & Output
res = 50;
y = linspace(0.01,0.99,res);
x = linspace(0.01,0.99,res);
s = zeros(res,res);
```

```
for i = 1:res
    for j = 1:res
        for k = 1:n
%               in = inpolygon(x(i),y(j),xb,yb);
            if in
                F = findf(x(i), y(j), xb(k), yb(k), nx(k), ny(k
                   ),lm(k));
                G = findg(x(i), y(j), xb(k), yb(k), nx(k), ny(k
                   ),lm(k));
                s(j,i) = s(j,i) + u(k)*G - q(k)*F;
            else
                s(j,i) = nan;
            end
        end
    end
end

[xgr,ygr] = meshgrid(x,y); zz = xgr + 1i*ygr;
true_sol = real(exp(zz));
% true_sol(inpolygon(xgr,ygr,xb,yb) == 0) = NaN;

figsize = [0,0,3,2.5];
fontsize = 9;
h = figure;
clf
surface(x,y,s-true_sol, 'edgecolor','none')
hold on
% graphics of elements
for i = 1:n
    xx = [xb(i),xb(i+ 1)];
    yy = [yb(i), yb(i + 1)];
    plot(xx, yy,'.-k','LineWidth',2,'MarkerSize',15)
end
% title ('Surface Plot for solution')
% xlabel ('x')
% ylabel('y')
% axis([0 1 0 1])
axis equal
colorbar
```

```
xlim([0 1]); ylim([0 1]);
set(h, 'Units','Inches');
pos = get(h,'Position');
set(h,...
    'Position', figsize,...
    'Units', 'Inches', ...
    'PaperPositionMode','Auto',...
    'PaperUnits','Inches')
set(gca, 'Units', 'normalized',...
    'FontUnits', 'points',...
    'FontWeight', 'normal',...
    'FontSize', fontsize, ...
    'FontName', 'Times',...
    'TickLabelInterpreter', 'latex');
print(h,'G:\My Drive\Projects\Kirk Project\Workings\report\
   figures\bem-error','-depsc')
```

FEM

This makes use of the MATLAB FEM PDE toolbox for creating the mesh and discretising the problem for solution.

```
model = createpde();
L = [2 6 0 2 2 1 1 0 0 0 1 1 2 2];
dl = decsg(L.');
geometryFromEdges(model, dl);
pdegplot(model,'EdgeLabels','on');
axis equal

clf
bc = @(location,state) real(exp(location.x + 1i*location.y));
applyBoundaryCondition(model,'dirichlet','Edge',1:model.
   Geometry.NumEdges,'r', bc);
specifyCoefficients(model,'m',0,'d',0,'c',-1,'a',0,'f',0);
hmax = 0.1;
generateMesh(model,'Hmax',hmax);

figsize = [0,0,2.5,2.5];
fontsize = 9;
h = figure;
```

```
pdemesh(model)
xlim([0 2]); ylim([0 2]);
set(h, 'Units','Inches');
pos = get(h,'Position');
set(h,...
    'Position', figsize,...
    'Units', 'Inches', ...
    'PaperPositionMode','Auto',...
    'PaperUnits','Inches')
set(gca, 'Units', 'normalized',...
    'FontUnits', 'points',...
    'FontWeight', 'normal',...
    'FontSize', fontsize, ...
    'FontName', 'Times',...
    'TickLabelInterpreter', 'latex');
print(h,'G:\My Drive\Projects\Kirk Project\Workings\report\
   figures\fem-mesh','-depsc')


results = solvepde(model);
u = results.NodalSolution;
pdeplot(model,'XYData',u)
title('Numerical Solution');
xlabel('x')
ylabel('y')

figsize = [0,0,3,2.5];
fontsize = 9;
h = figure;
p = model.Mesh.Nodes;
exact = real(exp(p(1,:) + 1i.*p(2,:)));
pdeplot(model, 'XYData', u - exact.');

xlim([0 2]); ylim([0 2]);
set(h, 'Units','Inches');
pos = get(h,'Position');
set(h,...
    'Position', figsize,...
    'Units', 'Inches', ...
    'PaperPositionMode','Auto',...
    'PaperUnits','Inches')
set(gca, 'Units', 'normalized',...
    'FontUnits', 'points',...
    'FontWeight', 'normal',...
    'FontSize', fontsize, ...
    'FontName', 'Times',...
    'TickLabelInterpreter', 'latex');
print(h,'G:\My Drive\Projects\Kirk Project\Workings\report\
   figures\fem-error','-depsc')

hmax = 0.1;
errs = [];
err = 1;
while err > 1e-8 % run until error <= 5e-7
    generateMesh(model,'Hmax',hmax); % refine mesh
    results = solvepde(model);
    u = results.NodalSolution;
    p = model.Mesh.Nodes;
    exact = real(exp(p(1,:) + 1i.*p(2,:)));
    err = norm(u - exact',inf); % compare with exact solution
    disp("err = " + err + " hmax = " +hmax);

    errs = [errs err]; % keep history of err
    hmax = hmax/2;
end

semilogy(  sqrt(1:numel(errs)), errs,'MarkerSize',12);
ax = gca;
ax.XTick = 1:numel(errs);
title('Error History');
xlabel('\sqrt{order}');
ylabel('Norm of Error');

p = model.Mesh.Nodes;
exact = (1 - p(1,:).^2 - p(2,:).^2)/4;
pdeplot(model,'XYData',u - exact')
title('Error');
xlabel('x')
ylabel('y')
```

Appendix B

Additional Figures

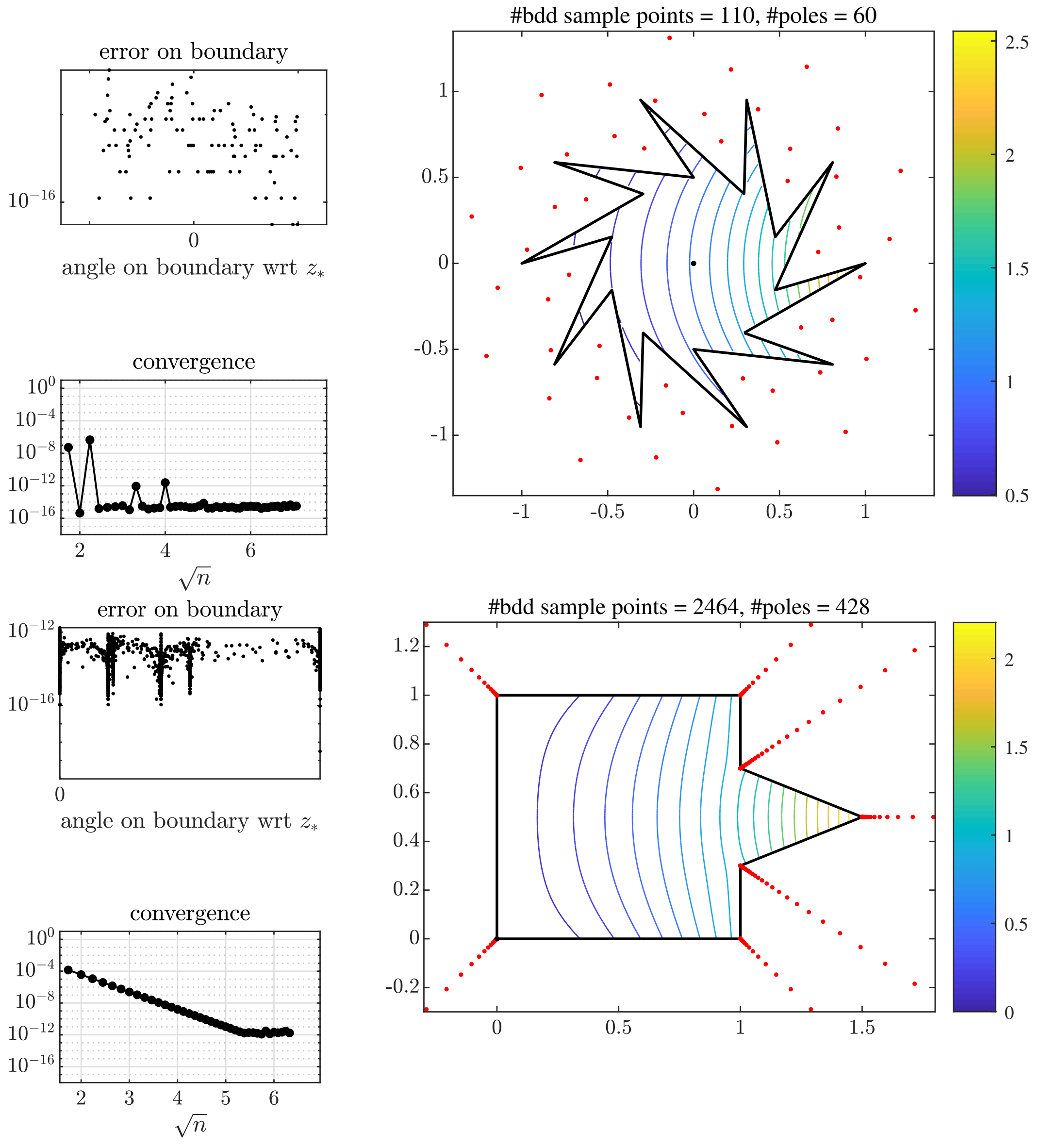

Figure B.1 $Re(exp(z))$ for the star, $Re(z)^2$ for the salient spike.

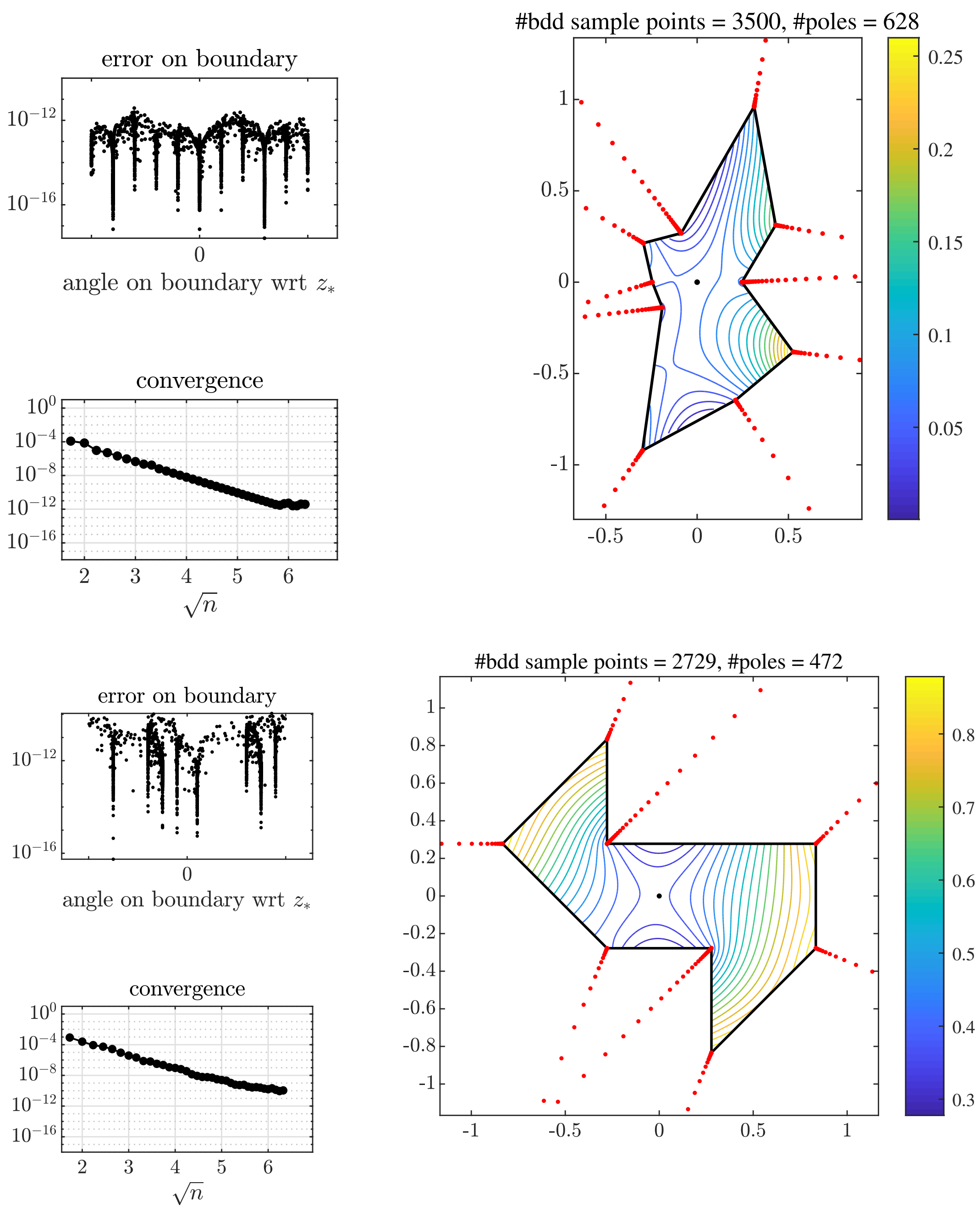

Figure B.2 $Re(z)^2$ for the random decagon, $|z|$ for the isospectral drum.

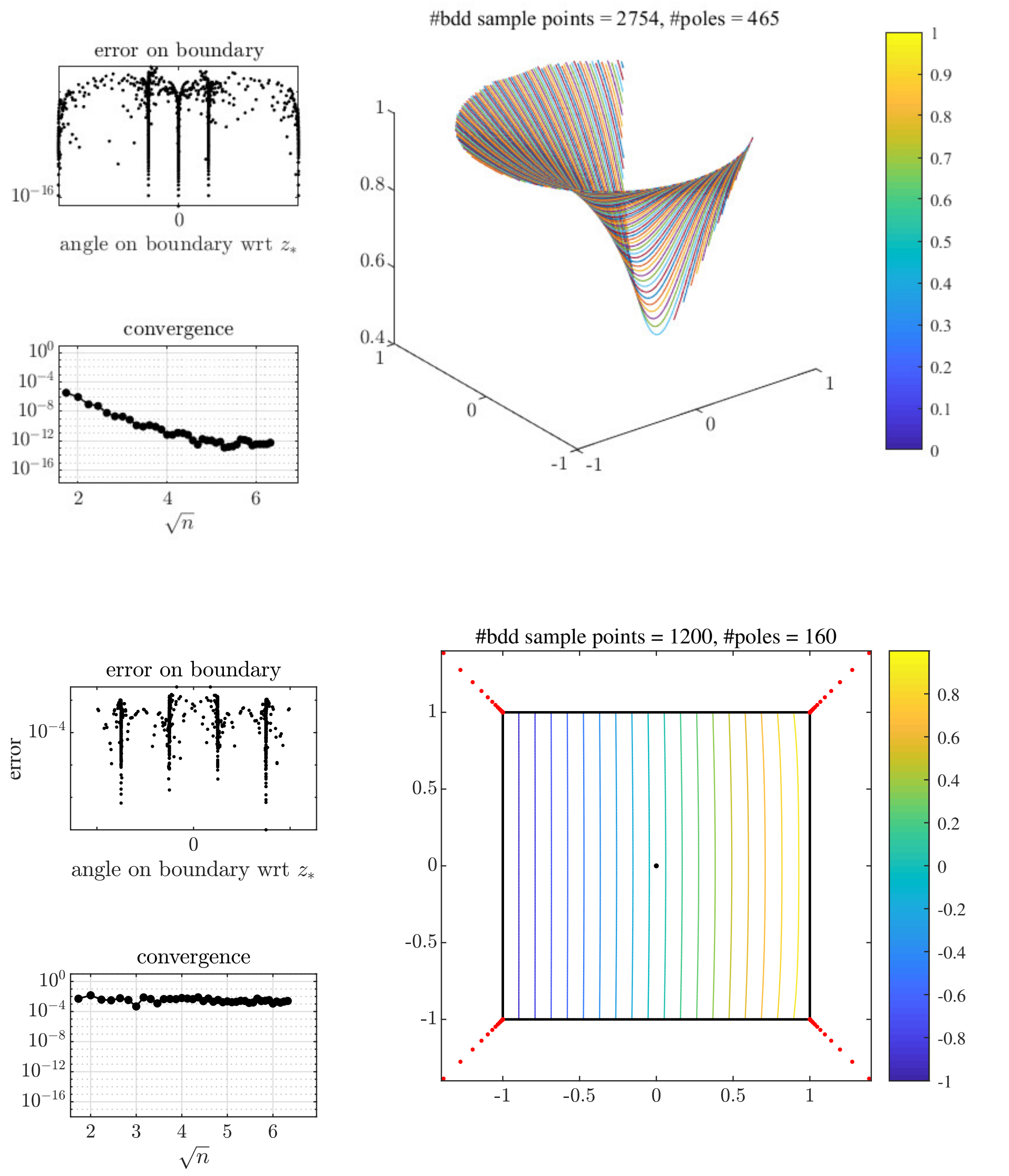

Figure B.3 $|z|$ for 3D view of the curved region, Neumann/Dirichlet of 1, $\Re(z), -1, \Re(z)$ for square. Note this is not convergent, not accurate.

References

[1] Henri Lebesgue. “Sur l’approximation des fonctions”. In: *Bull. Sci. Math* 22.10 (1898), pp. 11–20.

[2] Serge Bernstein. *Sur l’ordre de la meilleure approximation des fonctions continues par des polynômes de degré donné.* Vol. 4. Hayez, imprimeur des académies royales, 1912.

[3] AA Gončar. “On the rapidity of rational approximation of continuous functions with characteristic singularities”. In: *Mathematics of the USSR-Sbornik* 2.4 (1967), p. 561.

[4] Andrei Aleksandrovich Gončar. “Estimates of the growth of rational functions and some of their applications”. In: *Matematicheskii Sbornik* 114.3 (1967), pp. 489–503.

[5] J Szabados. “Uniform Approximation of Continuous Functions by Rational Functions”. PhD thesis. Ph. D. Thesis, Hungarian Academy of Sciences, 1968.(Hungarian), 1968.

[6] WJ Cody, G Meinardus, and RS Varga. “Chebyshev rational approximations to e^{-x} in $[0, +\infty)$ and applications to heat-conduction problems”. In: *Journal of Approximation Theory* 2.1 (1969), pp. 50–65.

[7] Eli Passow. “Another proof of Jackson’s theorem”. In: *Journal of Approximation Theory* 3.2 (1970), pp. 146–148.

[8] A Schönhage. “Zur rationalen Approximierbarkeit von e- xüber $[0, \infty)$”. In: *Journal of Approximation Theory* 7.4 (1973), pp. 395–398.

[9] Peter Henrici. “Applied and computational complex analysis, Volume 1”. In: (1974).

[10] Donald J Newman. “Rational approximation to e^{-x}”. In: *Journal of Approximation Theory* 10.4 (1974), pp. 301–303.

[11] Frank Stenger. “Explicit, nearly optimal, linear rational approximation with preassigned poles”. In: *mathematics of computation* 47.175 (1986), pp. 225–252.

[12] L.V. Ahlfors. *Complex Analysis: An Introduction to the Theory of Analytic Functions of One Complex Variable.* International series in pure and applied mathematics. McGraw-Hill Book Company, 1987.

[13] Herbert Stahl. "Best uniform rational approximation of $|x|$ on $[-1, 1]$". In: *Matematicheskii Sbornik* 183.8 (1992), pp. 85–118.

[14] G.H. Golub et al. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996, pp. 248–250. ISBN: 9780801854149.

[15] Herbert R Stahl et al. "Best uniform rational approximation of x^α on $[0, 1]$". In: *Acta mathematica* 190.2 (2003), pp. 241–306.

[16] Keng-Cheng Ang. "Introducing the boundary element method with MATLAB". In: *International Journal of Mathematical Education in Science and Technology* 39.4 (2008), pp. 505–519.

[17] Pedro Gonnet, Ricardo Pachón, and Lloyd N Trefethen. "Robust rational interpolation and least-squares". In: *Electron. Trans. Numer. Anal* 38 (2011), pp. 146–167.

[18] Ricardo Pachón, Pedro Gonnet, and Joris Van Deun. "Fast and stable rational interpolation in roots of unity and Chebyshev points". In: *SIAM Journal on Numerical Analysis* 50.3 (2012), pp. 1713–1734.

[19] Pedro Gonnet, Stefan Guttel, and Lloyd N Trefethen. "Robust Padé approximation via SVD". In: *SIAM review* 55.1 (2013), pp. 101–117.

[20] Abinand Gopal and Lloyd N Trefethen. "Brief Report: New Laplace and Helmholtz solvers". In: *Proceedings of the National Academy of Sciences of the United States of America* 116.21 (2019), p. 10223.

[21] Abinand Gopal and Lloyd N Trefethen. "Solving Laplace problems with corner singularities via rational functions". In: *SIAM Journal on Numerical Analysis* 57.5 (2019). Implementation available at https://people.maths.ox.ac.uk/trefethen/lightning.html, pp. 2074–2094.

[22] Daan Huybrechs and Anda-Elena Olteanu. "An oversampled collocation approach of the wave based method for Helmholtz problems". In: *Wave Motion* 87 (2019), pp. 92–105.

[23] Lloyd N Trefethen. *Approximation theory and approximation practice*. Vol. 164. Siam, 2019.

[24] Lloyd N Trefethen. "Numerical conformal mapping with rational functions". In: *arXiv preprint arXiv:1911.03696* (2019).

[25] Stefano Costa. "Solving Laplace problems with the AAA algorithm". In: *arXiv preprint arXiv:2001.09439* (2020).

[26] Jeremy Hoskins and Manas Rachh. "On the discretization of Laplace's equation with Neumann boundary conditions on polygonal domains". In: *arXiv preprint arXiv:2001.05434* (2020).