

Optimal Sampling for Linear Function Approximation and High-Order Finite
Difference Methods over Complex Regions

by

Tony Liu

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

Approved July 2019 by the
Graduate Supervisory Committee:

Rodrigo Platte, Chair
Rosemary Renaut
Mohamed Moustaoi
Sebastien Motsch
David Kaspar

ARIZONA STATE UNIVERSITY

August 2019

ABSTRACT

This dissertation focuses on algorithms that generate good sampling points for function approximation. In 1D, it is well known that polynomial interpolation using equispaced points is unstable. On the other hand, using Chebyshev nodes provides both stable and highly accurate points for polynomial interpolation. In higher dimensional complex regions, optimal sampling points are not known explicitly. This work presents robust algorithms that find good sampling points in complex regions for polynomial interpolation, least-squares, and radial basis function (RBF) methods. The quality of these nodes is measured using the Lebesgue constant. We will also consider optimal sampling for constrained optimization, used to solve PDEs, where boundary conditions must be imposed. Furthermore, we extend the scope of the problem to include finding near-optimal sampling points for high-order finite difference methods. These high-order finite difference methods can be implemented using either piecewise polynomials or RBFs.

TABLE OF CONTENTS

	Page
LIST OF TABLES	v
LIST OF FIGURES	vi
CHAPTER	
1 Introduction	1
1.1 Data Fitting Methods	1
1.2 High-Order Finite Difference Methods	2
1.3 Outline	3
2 Preliminaries	4
2.1 Polynomial Data Fitting Background	4
2.2 The Lebesgue Constant	7
2.3 Formulation in 2D	11
2.4 Conformal Mappings	14
3 Optimal Sampling for Function Approximation Methods	16
3.1 Optimal Points for Polynomial Data Fitting Methods	16
3.1.1 Computational Algorithms for Finding Optimal Points	16
3.1.2 Numerical Experiments on Complex 2D Regions	20
3.1.3 Saving Computation Time	23
3.2 Optimal Points for Constrained Approximation	30
3.2.1 Algorithmic Considerations	32
3.2.2 Sampling Near-optimal Points using Approximate-Fekete Points	33
3.2.3 Comparing the Greedy Algorithm and the Column-Pivoting	
QR Algorithm	36
3.2.4 Modifying the Column-Pivoting QR Algorithm for Constrained	
Approximation	38

CHAPTER	Page
3.2.5	Results 40
3.3	Optimal Sampling for RBF Interpolation 45
3.3.1	RBF Methods Background 45
3.3.2	Finding Optimal Points for RBF Interpolation 47
3.3.3	Results in 2D 48
3.3.4	Results for Surfaces and 3D Regions 49
4	Sampling Nodes for High-Order Finite Difference Methods 54
4.1	Polynomial Finite Difference Methods 54
4.1.1	Background 54
4.1.2	Piecewise Polynomial Formulation 56
4.1.3	Partial Centered Finite Difference Dimension Reduction Method 59
4.1.4	KTE Dimension Reduction Method 63
4.1.5	Column-Pivoting Method 66
4.2	RBF Finite Difference (RBF-FD) Methods in 1D 67
4.3	RBF-FD Methods in 2D 69
4.3.1	RBF-FD Methods with Polyharmonic Splines and Polynomials Background 69
4.3.2	Applying the Column-Pivoting QR Method 75
4.3.3	Results in 2D 76
4.3.4	Solving PDEs Using RBF-FD Methods with Polyharmonic Splines and Polynomials 81
5	Final Remarks 83
5.1	Ongoing Work 83
5.2	Conclusions 84

CHAPTER	Page
REFERENCES	86

LIST OF TABLES

Table	Page
2.1 Monomial and Chebyshev Basis Functions	4
3.1 Chebyshev Basis Functions in 2D	20
3.2 Lebesgue Constants for Approximate-Fekete Points and Padua Points .	36
3.3 RBF Examples	46

LIST OF FIGURES

Figure	Page
2.1 Lagrange Polynomials Example	6
2.2 Runge Function Interpolation	8
2.3 Lebesgue Function Example	9
2.4 L-shape Region	14
2.5 Conformal Mapping Points	15
3.1 Unit Circle Discretization	20
3.2 Complex 2D Regions	21
3.3 Peanut Region Greedy Algorithm Points	22
3.4 Bumped-disk Ring Region Greedy Algorithm Points	22
3.5 L-shape Region Greedy Algorithm Points	23
3.6 Approximate-Fekete Points	35
3.7 Square Region Padua Points	36
3.8 Algorithm Comparison for the Least Squares Problem	38
3.9 Algorithm Comparison for the Interpolation Problem	38
3.10 Optimal Sampling Points for Constrained Approximation	41
3.11 Solution Convergence on the Circle Region	42
3.12 Solution Node Refinement Convergence on the Square Region	43
3.13 Solution Node Refinement Convergence on the Peanut Region	44
3.14 Solution Node Refinement Convergence on the Bumped-disk Region ...	45
3.15 Gaussian RBF centered at the origin, $\xi = 2$	46
3.16 Optimal Samplings Points for RBF Interpolation on the Peanut	49
3.17 Equispaced Points for RBF Interpolation on the Peanut	49
3.18 Lebesgue Function for Peanut Region Points	50
3.19 Optimal Samplings Points for RBF Interpolation on the Bumped-disk .	50

Figure	Page
3.20	Equispaced Points for RBF Interpolation on the Bumped-disk 51
3.21	Lebesgue Function for Bumped-disk Region Points 51
3.22	Lebesgue Function for Half-Torus Surface Points 52
3.23	Lebesgue Function for Hyperboloid Surface Points 53
3.24	Hyperboloid 3D Sampling Points 53
4.1	Piecewise Polynomial Stencil Grouping for $m = 4, k = 3$ 56
4.2	Lagrange Piecewise Polynomials 58
4.3	Lebesgue Function for Piecewise Polynomial Example 58
4.4	Differentiation Stencils 61
4.5	Partial Centered Finite Difference Method Example 1 62
4.6	Partial Centered Finite Difference Method Example 2 63
4.7	KTE Example 65
4.8	Optimal α Values for Various Nodal Set and Stencil Sizes 65
4.9	Finite Difference Column-pivoting QR Example 67
4.10	Finite Difference Column-pivoting QR Example with RBF Basis 70
4.11	Square Shape Equispaced and Hexagonal Node Layout 71
4.12	15 Node Stencil Example for the Bumped-disk Region 72
4.13	Convergence Rates of a First-Order Derivative Approximation using PHS and Polynomials 74
4.14	Dirichlet Eigenvalues for Approximation using PHS and Polynomials . . 75
4.15	Starting Guess Points and Optimal Samplings Points for the Bumped- disk Region 77
4.16	Differentiation Matrix Eigenvalues for Starting Guess Points and Op- timized Points 78

Figure	Page
4.17 Starting Guess Points and Optimal Samplings Points for the Peanut ...	78
4.18 Eigenvalue Improvement for Bumped-disk Region	79
4.19 Sampling Node Accuracy Comparison for the Bumped-disk and Peanut Region	80
4.20 Eigenvalue Improvement for Peanut Region	81
4.21 PDE Accuracy For RBF-FD Methods in 2D	82

Chapter 1

INTRODUCTION

1.1 Data Fitting Methods

In the case of 1D polynomial interpolation, such as on the interval $[-1, 1]$, it is well known that equispaced points provide unstable interpolation nodes. These nodes generate oscillations near the boundary, also known as the Runge phenomenon [39]. The solution to this stability problem is to use points that cluster around the edges of the domain, for example, Chebyshev points. These points are near-optimal for polynomial interpolation. As a result, near-optimal points on any interval of the real line can be obtained by scaling and translating Chebyshev points.

Finding optimal sampling points in the multivariate case proves to be a much more difficult task. Some unique point sets that provide good approximations exist; however, these are only known for specific regions. One example is the tensor-product Chebyshev points for the unit square region. Multi-dimensional regions present a more complicated problem when searching for optimal points due to the fact that these regions can take on complex forms that cannot be handled in the same manner as we handle intervals of the real line or simple 2D regions. This motivates the need to develop a robust algorithm for finding sampling points for stable approximations on bounded domains in 2D and higher dimensions.

One way to determine the optimality of a given set of sampling points is by investigating the Lebesgue constant. The Lebesgue constant provides a bound for the interpolation error relative to the error of the best approximation [39]. In this work, we develop our methods for finding good sample points by constraining the

growth of the Lebesgue constant.

Finding points for interpolation in general, to include the 1D case, has been heavily studied. Recent papers [5, 6, 7, 8, 9, 19, 20, 28, 29, 38, 40] have presented algorithms for constructing optimal points for polynomial data fitting. Each of these algorithms is centered around optimizing a certain objective function which bounds the interpolation error. The algorithms presented in this work provide robust methods for polynomial data fitting on general complex 2D regions. This work includes speed-up methods which allow us to generate optimal points to a higher degree than what is currently presented in literature.

Additionally, we also provide algorithmic considerations that allow us to find optimal sampling points for complex regions for constrained approximation. Specifically, we consider approximation with specified boundary constraints in order to solve PDEs. Lastly, we also consider sampling points for RBF data fitting methods.

It is important to note that the sampling points obtained from this work are near-optimal. The optimization problem introduced here is known to be NP-hard [6]. The algorithms provided in this work are greedy procedures and not always optimal, but compute points which provide Lebesgue constants that are sufficiently small for stable approximations. The optimality trade-off enables us to reduce the computational challenges of the problem. Any mention of optimal sampling points resulting from this work should be regarded as near-optimal.

1.2 High-Order Finite Difference Methods

This work will also present methods for finding optimal sampling points for high-order finite difference methods in both 1D and 2D. We formulate the finite difference methods using piecewise polynomials and Radial Basis Functions (RBFs). Previously developed methods for finding optimal sampling points for finite difference methods

can be found in [23]. The methods mentioned in [23] also formulate the finite difference methods using piecewise polynomials. For the finite difference approach, we again investigate the optimality of sampling points using the Lebesgue constant. In the 1D case, we are able to formulate a constrained optimization problem and obtain optimal sampling points using a global minimum finder (MultiStart from MATLAB's Optimization Toolbox [1]). This formulation is computationally feasible due to various dimension reduction methods we implement.

The complexities of 2D regions once again complicates the process of finding optimal sampling points in the case of high-order finite difference methods. The structural properties we leverage to be able to solve a global optimization problem in 1D in a reasonable amount of time are not present in complex 2D regions. As a result, we implement a well-known algorithm that uses QR factorization with column-pivoting to find optimal sampling points.

1.3 Outline

The remainder of this thesis will be organized in the following manner. In Section 2, we briefly discuss the background relevant to formulating the problem of finding optimal sampling points for data fitting. Section 3 presents the algorithms we implement to obtain the optimal sampling points for data fitting and discusses results for complex 2D regions. We discuss two methods that we implement to speed up the computation of the algorithms outlined. Further, we also discuss algorithmic considerations and numerical examples for constrained approximation and RBF approximation. Section 4 presents the material regarding the sampling of optimal points for high-order finite difference methods using both piecewise polynomials and RBFs. Results are discussed for both the 1D and the 2D cases. Lastly, Section 5 concludes the results of this work and discusses possible future research directions.

Chapter 2

PRELIMINARIES

2.1 Polynomial Data Fitting Background

In many applications, it is desirable to approximate and infer processes which can only be sampled or measured at specific locations. The most commonly implemented solution to this problem is interpolation. More precisely, given the $n+1$ pairs, (x_j, y_j) , the problem consists of determining an approximating function, $p(x)$, which satisfies $p(x_j) = y_j$. One way to do this is to assume a linear form for all interpolating functions. The problem is then formulated as:

$$p(x) = \sum_{j=0}^n c_j \phi_j(x). \quad (2.1)$$

The c_j 's are the unknown coefficients while $\phi_j(x)$ are the preselected basis functions. Two examples of such basis functions are the monomial basis and the Chebyshev basis which are defined below in Table 2.1.

	Monomial	Chebyshev
$\phi_0(x)$	1	1
$\phi_1(x)$	x	$\cos(\arccos x)$
$\phi_2(x)$	x^2	$\cos(2 \arccos x)$
$\vdots$	$\vdots$	$\vdots$
$\phi_n(x)$	x^n	$\cos(n \arccos x)$

Table 2.1: Monomial and Chebyshev Basis Functions

We can then represent the interpolation problem as the following linear system:

$$\begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \dots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \phi_2(x_n) & \dots & \phi_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad (2.2)$$

The matrix in the left hand side of Equation 2.2 is referred to the Vandermonde matrix and can be denoted by:

$$V_n[\mathbf{x}] = V_n[(x_0, x_1, \dots, x_n)] = \begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \dots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \phi_2(x_n) & \dots & \phi_n(x_n) \end{bmatrix} \quad (2.3)$$

where $\mathbf{x}$ is the vector that contains the sampling points and n represents the degree of the approximation. In the case where the Vandermonde matrix is square, we have approximation is called interpolation, whereas if the matrix is overdetermined, we have a least-squares problem. The existence and uniqueness of polynomial interpolants in 1D has been studied and is summarized in Theorem 2.4 [39].

Theorem 2.4 *Given $n+1$ distinct data points $(x_0, y_0) \dots (x_n, y_n)$, there exists a unique polynomial $p(x)$ such that $p(x_j) = y_j$ for $j = 0, \dots, n$.*

Another important formulation of the interpolant comes from the Lagrange basis: $p(x) = \sum_{j=0}^n y_j l_j(x)$. In this formulation, l_j represents the Lagrange polynomial basis and the y_j represents the sampled data. The Lagrange polynomial basis is obtained from:

$$l_j(x_k) = \begin{cases} 1 & k = j \\ 0 & k \neq j \end{cases} \quad (2.5)$$

$$l_j(x) = \frac{\prod_{k \neq j} (x - x_k)}{\prod_{k \neq j} (x_j - x_k)} \quad (2.6)$$

This formulation for $p(x)$ will be required in order to calculate the Lebesgue constant [39]. Figure 2.1, for example, plots two Lagrange polynomials defined using 20 sampling points.

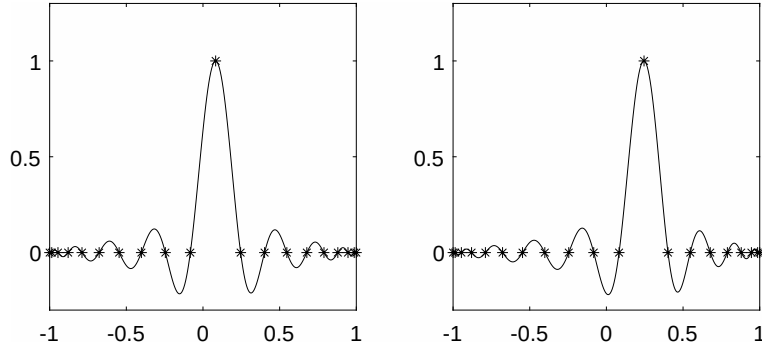


Figure 2.1: Lagrange Polynomials. *Left:* $l_{10}(x)$. *Right:* $l_{11}(x)$

We notice that the j -th Lagrange polynomial has value 1 at x_j and 0 at all other interpolation points. Then, to evaluate the interpolant at any point in the domain $[-1, 1]$ we simply evaluate: $p(x) = \sum_{j=0}^{19} y_j l_j(x)$. Another way to formulate Lagrange functions is obtained from the determinant of the Vandermonde matrix in Equation 2.3:

$$l_j(x) = \frac{\det(V_n[(x_0, x_1, \dots, x_{j-1}, x, x_{j+1}, \dots, x_n)])}{\det(V_n[x_0, \dots, x_n])}. \quad (2.7)$$

The error that results from interpolating a given function using polynomials at nodes $x_0, \dots, x_n$ is given by the following theorem.

Theorem 2.8 *Let $x_0, \dots, x_n$ be $n+1$ distinct nodes and let x be a point belonging to the domain of a given function. Assume that $f \in C^{n+1}(I_x)$, where I_x is the smallest interval containing the nodes $x_0, \dots, x_n$. Then the interpolation error at the point x is given by*

$$E_n(x) = f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \omega_{n+1}(x)$$

where $\xi \in I_x$ and ω_{n+1} is the nodal polynomial defined as $\omega_{n+1} = \prod_{j=0}^n (x - x_j)$.

One important result, is the relationship between Chebyshev points and the interpolation error. Theorem 2.9 summarizes these results. It is shown that the Chebyshev points minimize the nodal polynomial defined in Theorem 2.8.

Theorem 2.9 *For $n \geq 0$, let $x_0, x_1, \dots, x_n \in \mathbb{R}$ and set $\omega_{n+1}(x) = \prod_{j=0}^n (x - x_j)$. Then*

$$\sup_{a \leq x \leq b} |\omega_{n+1}(x)| \geq 2^{-n}$$

and if $x_i = \cos \frac{(2i+1)\pi}{(2n+2)}$, then

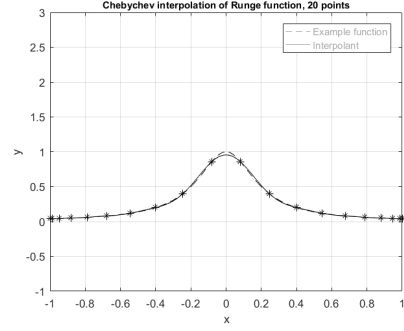
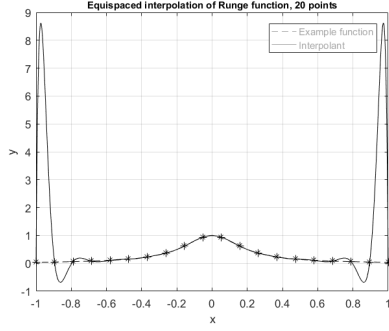
$$\sup_{a \leq x \leq b} |\omega_{n+1}(x)| = 2^{-n}$$

The points defined by $x_i = \cos \frac{(2i+1)\pi}{(2n+2)}$ are known as the Chebyshev points of the first kind.

2.2 The Lebesgue Constant

As mentioned previously, in the 1D case, equispaced points are unstable and oscillate near the boundary of the domain. Chebyshev points provide near-optimal sampling points and lead to highly accurate and stable approximations. This can be seen in the Runge example pictured in Figure 2.2.

In the case of Chebyshev points, the approximation no longer diverges near the endpoints. It is important to quantify the divergent behavior in order to determine



(a) Interpolant using 20 equispaced points (b) Interpolant with 20 Chebyshev points

Figure 2.2: Runge Function, $f(x) = \frac{1}{1+25x^2}$, interpolants

the optimality of sampling points. This is where the Lebesgue constant becomes important. The Lebesgue constant is defined as:

$$\Lambda = \sup_f \frac{\|p_f\|_\infty}{\|f\|_\infty}.$$

We note that the norm in the numerator is a sup-norm over the domain of the interpolant for f, p_f , while the norm in the denominator may be a max norm over a set of discrete sampled data points. One way to interpret the Lebesgue constant is: if the function we are trying to approximate has value no greater than 1 on our domain $([-1, 1])$, what is the largest possible value of our interpolant on the same domain. Another interpretation is: how much will our interpolant be perturbed if the data samples we have for the function to be approximated include noise?

Theorem 2.10 establishes an important relationship between the best polynomial interpolant and the Lebesgue constant [39].

Theorem 2.10 *Let Λ be the Lebesgue constant for a linear projection L of $C([-1, 1])$ onto the polynomial space P_n . Let f be a function in $C([-1, 1])$, $p = LF$ the corresponding polynomial interpolant to f , and p^* the best approximation. Then*

$$\|f - p\| \leq (1 + \Lambda) \|f - p^*\|.$$

To obtain the Lebesgue constant, we first formulate the Lebesgue function: $\lambda(x) = \sum_{j=0}^n |l_j(x)|$ where $l_j(x)$ comes from the Lagrange formulation. Then, the Lebesgue constant is defined to be $\Lambda = \sup_{x \in [-1,1]} \lambda(x)$. We should note that the Lebesgue constant can be define for any linear approximation method. Thus if we happen to change the basis, or use a different approximation method, we can still calculate the Lebesgue constant as long as we can calculate the Lagrange polynomials.

Looking at the Lebesgue constants in the previous Runge example (Figure 2.2), we see that it captures the divergent behavior. This is pictured in Figure 2.3.

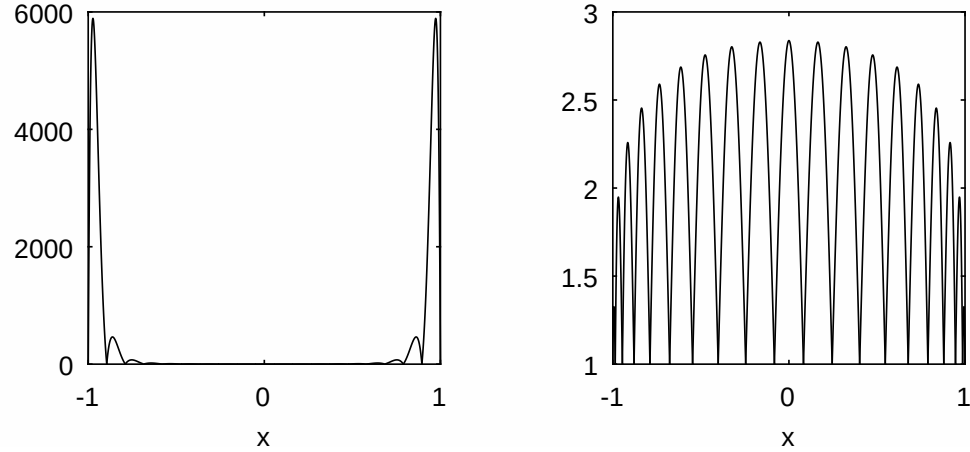


Figure 2.3: *Left:* Lebesgue function using 20 equispaced points. *Right:* Lebesgue function using 20 Chebyshev points.

Theorem 2.11 [39] summarizes the asymptotic behavior of Lebesgue constants with respect to interpolation points. This gives us a good intuition on how interpolants perform based on the selection of sampling nodes as we increase the dimension of interpolation.

Theorem 2.11 *The Lebesgue constants Λ_n for degree $n \geq 0$ polynomial interpolation in any set of $n + 1$ distinct points in $[-1, 1]$ satisfy*

$$\Lambda_n \geq \frac{2}{\pi} \log(n+1) + \frac{2}{\pi} \left(\gamma + \log\left(\frac{4}{\pi}\right) \right),$$

where $\gamma \approx .577$ is Euler's constant.

For Chebyshev points, they satisfy

$$\Lambda_n \leq \frac{2}{\pi} \log(n+1) + 1, \quad \Lambda_n \sim \frac{2}{\pi} \log(n), \quad n \rightarrow \infty.$$

For equispaced points, they satisfy

$$\Lambda_n > \frac{2^{n-2}}{n^2}, \quad \Lambda_n \sim \frac{2^{n+1}}{n \log(n)}, \quad n \rightarrow \infty.$$

Theorem 2.11 shows that the Lebesgue constants for Chebyshev points grow at most logarithmically while the Lebesgue constants for equispaced points grow exponentially. This demonstrates the unstable nature of interpolating with equispaced sampling points and why Chebyshev points produce more accurate interpolation results.

Formally, we can also view the Lebesgue constant as the condition number of the interpolation process. Consider the absolute condition number defined as

$$\limsup_{\epsilon \rightarrow 0, |\delta| < \epsilon} \frac{\|F(f + \delta) - F(f)\|}{\|\delta\|} = \limsup_{\epsilon \rightarrow 0, |\delta| < \epsilon} \frac{\|\sum_i l_i(x)(f_i + \delta_i) - \sum_i l_i(x)f_i\|_\infty}{\max_j |\delta_j|} \quad (2.12)$$

$$= \limsup_{\epsilon \rightarrow 0, |\delta| < \epsilon} \frac{\|\sum_i l_i(x)(\delta_i)\|_\infty}{\max_j |\delta_j|} \quad (2.13)$$

$$= \limsup_{\epsilon \rightarrow 0, |\delta| < \epsilon} \frac{\max_{-1 \leq x \leq 1} |\sum_i l_i(x)(\delta_i)|}{\max_j |\delta_j|} \quad (2.14)$$

$$\leq \limsup_{\epsilon \rightarrow 0, |\delta| < \epsilon} \frac{\max_{-1 \leq x \leq 1} \sum_i |l_i(x)| |\delta_i|}{\max_j |\delta_j|} \quad (2.15)$$

The equality is realized when the sign of δ_i and $l_i(x)$ match. That is, if $\text{sgn}(\delta_i) = \text{sgn}(l_i(x))$, then

$$\limsup_{\epsilon \rightarrow 0, |\delta| < \epsilon} \frac{\|F(f + \delta) - F(f)\|}{\|\delta\|} = \limsup_{\epsilon \rightarrow 0, |\delta| < \epsilon} \frac{\max_{-1 \leq x \leq 1} \sum_i |l_i(x)| |\delta_i|}{\max_j |\delta_j|} \quad (2.16)$$

$$= \max_{-1 \leq x \leq 1} \sum_i |l_i(x)| \quad (2.17)$$

$$= \Lambda. \quad (2.18)$$

Thus, we see that the Lebesgue constant can be viewed as the condition number of the interpolation process.

2.3 Formulation in 2D

In 2D, we consider sampling points, (x_j, y_j) . The problem then consists of determining an approximating function, $p(x, y)$, which satisfies $p(x_j, y_j) = z_j$. Again, we can assume a linear form for interpolating functions. The problem is formulated as:

$$p(x, y) = \sum_{j=0}^n c_j \phi_j(x_j, y_j). \quad (2.19)$$

In 2D, the resulting linear interpolation system takes the form:

$$\begin{bmatrix} \phi_0(x_0, y_0) & \phi_1(x_0, y_0) & \phi_2(x_0, y_0) & \phi_3(x_0, y_0) & \dots & \phi_n(x_0, y_0) \\ \phi_0(x_1, y_1) & \phi_1(x_1, y_1) & \phi_2(x_1, y_1) & \phi_3(x_1, y_1) & \dots & \phi_n(x_1, y_1) \\ \phi_0(x_2, y_2) & \phi_1(x_2, y_2) & \phi_2(x_2, y_2) & \phi_3(x_2, y_2) & \dots & \phi_n(x_2, y_2) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n, y_n) & \phi_1(x_n, y_n) & \phi_2(x_n, y_n) & \phi_3(x_n, y_n) & \dots & \phi_n(x_n, y_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

To investigate the optimality of a given set of nodes, we generate the Lebesgue function. In order to do this, we determine the Lagrange polynomials l_j . The formulation in 2D takes is defined as:

$$l_j(x_k, y_k) = \begin{cases} 1 & k = j \\ 0 & k \neq j. \end{cases} \quad (2.20)$$

This can also be expressed as a linear system:

$$\mathbf{A}\mathbf{G} = \mathbf{I} \quad (2.21)$$

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} \phi_0(x_0, y_0) & \phi_1(x_0, y_0) & \phi_2(x_0, y_0) & \phi_3(x_0, y_0) & \dots & \phi_n(x_0, y_0) \\ \phi_0(x_1, y_1) & \phi_1(x_1, y_1) & \phi_2(x_1, y_1) & \phi_3(x_1, y_1) & \dots & \phi_n(x_1, y_1) \\ \phi_0(x_2, y_2) & \phi_1(x_2, y_2) & \phi_2(x_2, y_2) & \phi_3(x_2, y_2) & \dots & \phi_n(x_2, y_2) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n, y_n) & \phi_1(x_n, y_n) & \phi_2(x_n, y_n) & \phi_3(x_n, y_n) & \dots & \phi_n(x_n, y_n) \end{bmatrix} \\
\mathbf{G} &= \begin{bmatrix} g_{0,0} & g_{0,1} & g_{0,2} & g_{0,3} & \dots & g_{0,n} \\ g_{1,0} & g_{1,1} & g_{1,2} & g_{1,3} & \dots & g_{1,n} \\ g_{2,0} & g_{2,1} & g_{2,2} & g_{2,3} & \dots & g_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n,0} & g_{n,1} & g_{n,2} & g_{n,3} & \dots & g_{n,n} \end{bmatrix} \\
\mathbf{I} &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}
\end{aligned}$$

The Lagrange polynomials are then formulated as:

$$l_j(x_k, y_k) = g_{0,j}\phi_0(x_k, y_k) + g_{1,j}\phi_1(x_k, y_k) + \dots + g_{n,j}\phi_n(x_k, y_k). \quad (2.22)$$

It becomes clear that the Lagrange polynomial formulation that results from the $g_{j,k}$ coefficients of the linear system in Equation 2.21 satisfies the requirement in Equation

2.20. The matrix that gives us our l_j coefficients is:

$$\mathbf{G} = \begin{bmatrix} \phi_0(x_0, y_0) & \phi_1(x_0, y_0) & \phi_2(x_0, y_0) & \phi_3(x_0, y_0) & \dots & \phi_n(x_0, y_0) \\ \phi_0(x_0, y_0) & \phi_1(x_1, y_1) & \phi_2(x_1, y_1) & \phi_3(x_1, y_1) & \dots & \phi_n(x_1, y_1) \\ \phi_0(x_0, y_0) & \phi_1(x_2, y_2) & \phi_2(x_2, y_2) & \phi_3(x_2, y_2) & \dots & \phi_n(x_2, y_2) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_0, y_0) & \phi_1(x_n, y_n) & \phi_2(x_n, y_n) & \phi_3(x_n, y_n) & \dots & \phi_n(x_n, y_n) \end{bmatrix}^{-1} = \mathbf{A}^{-1} \quad (2.23)$$

The determinant formulation in Equations 2.7 has led to numerous criteria for finding optimal sampling points in 2D regions. For example, in [6, 8], sampling points in 2D regions are determined by finding the points which maximize the denominator term in Equation 2.7. In [8], these sampling points (also known as the approximate-Fekete points) are found for simple 2D regions such as the triangle, square, and circle while [6] computes approximate-Fekete points for more complex asymmetric 2D regions. In this setting, additional sampling points (called Leja points) are computed at each iteration by choosing the point in the domain where the maximum of the denominator term in Equation 2.7 occurs. The approximate-Fekete points are found using a column-pivoting QR algorithm. This algorithm is very robust for 2D geometries; however, it does not directly aim to minimize the Lebesgue constant while the optimal sampling method we discuss in Section 3 does.

One thing to note is that in order to find the optimal sampling points from Equations 2.21 - 2.23, we must be able to invert the matrix $\mathbf{A}$. We present the methods we implement to ensure that $\mathbf{A}$ is nonsingular in Chapter 3 along with the algorithms we use to find the optimal sampling points.

2.4 Conformal Mappings

It is reasonable to think that optimal sampling points for complex 2D regions could be obtained by mapping points from one region into another. That is, suppose we know the locations for a near-optimal set of sampling nodes for a simple 2D region. We could then find optimal sampling points for similar regions by finding a mapping between the two shapes and mapping the optimal sampling points for the simple shape onto the more complex 2D shape.

For example, suppose we wish to find optimal sampling points for the L-shape region shown in Figure 2.4 below. The Chebyshev tensor-product points for the Square shape are known to be near-optimal. For instance, 100 Chebyshev tensor-product points for the Square has a Lebesgue constant of $\Lambda = 22.28$. In Figure 2.5 below, we plot the Chebyshev tensor-product points as well as two conformal mappings onto the L-shape region. The mappings are obtained using the Schwarz-Christoffel Toolbox for MATLAB. More information on these mappings can be found in [12].

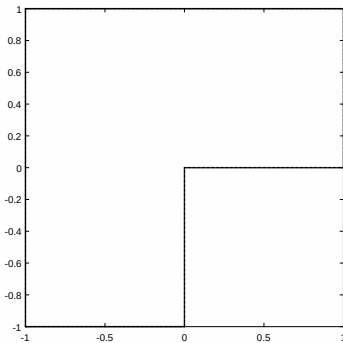
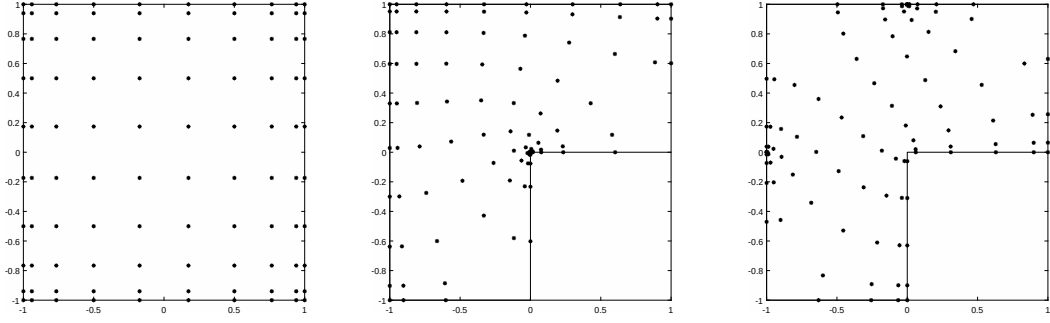


Figure 2.4: L-shape Region

The first conformal mapping takes corner of the square $(1, -1)$ and maps it to $(0, 0)$. The rest of the corners are mapped to the same location on the L-shape region. In the second conformal mapping, we map $(-1, 1)$ to $(-1, 0)$, $(1, 1)$ to $(0, 1)$, $(-1, -1)$ to $(0, -1)$, and $(1, -1)$ to $(1, 0)$. The resulting Lebesgue constant for first

conformal mapping is $\Lambda = 4.7 \times 10^3$, while the Lebesgue constant for second conformal mapping is $\Lambda = 2.06 \times 10^3$. Thus, we see that conformal mappings with sampling points from one shape to another does not preserve optimality. This motivates the need for a robust algorithm for finding near-optimal sampling points on complex 2D regions.



(a) Tensor-product Cheby- (b) Conformal Mapping 1 (c) Conformal Mapping 2
shev Points

Figure 2.5: Chebyshev Tensor-product Points and Conformal Mapping onto L-shape Region

OPTIMAL SAMPLING FOR FUNCTION APPROXIMATION METHODS

3.1 Optimal Points for Polynomial Data Fitting Methods

3.1.1 Computational Algorithms for Finding Optimal Points

In this chapter, we study a greedy algorithm for minimizing the Lebesgue constant. In this process, we begin with an initial set of sampling points and add sampling points one at a time, thus creating nested sets of points at every iteration. Specifically, given an initial set of sampling nodes, $(x_j)_{j=0}^n$, find and add the point x_{n+1} that satisfies

$$x_{n+1} = \operatorname{argmax}_{[-1,1]} \lambda(x)$$

to generate the new nested set $(x_j)_{j=0}^n \cup x_{n+1}$. This process is detailed in Algorithm 1 below.

Algorithm 1 Greedy Algorithm

1. Start with given set of interpolation points: $(x_j)_{j=0}^n$
 2. For the current set of interpolation points, calculate the Lebesgue function:
 $\lambda(x) = \sum_{j=0}^n |l_j(x)|$
 3. Find the maximum of the Lebesgue function
 4. Add new interpolation point at the location of the maximum: $x_{n+1} = \operatorname{argmax}_{[-1,1]} \lambda(x)$
 5. Iterate until the desired number of interpolation points is reached
-

Many previous works have implemented similar greedy algorithms in order to find optimal sampling nodes. The Leja points, as mentioned in Section 2.3, is one

example of such a greedy algorithm with the deviation that these points are found using a different objective function. In the process for finding Leja points, a greedy algorithm is applied to place a new point, x_{n+1} satisfying

$$x_{n+1} = \operatorname{argmax}_{x \in [-1,1]} |\det(V_{n+1}(x_0, \dots, x_n, x))|$$

where $V_{n+1}(x_0, \dots, x_n, x)$ follows the definition in Equation 2.7.

Other greedy algorithms can be found in [29, 40]. In [29], a greedy algorithm which computes optimal sampling points for the Square and Circle regions is presented. The greedy algorithm in this case uses an objective function formulated by taking the 2-norm of weighted Lagrange polynomials. In [40], the greedy algorithm uses a Lebesgue constant approximation to add new points. The approximation takes the Frobenious norm of the Lagrange polynomials rather than the max norm of the Lagrange polynomials and is done to save computation time as for high degree, calculating the Lebesgue constant becomes taxing computationally. Reference [40] also presents a greedy algorithm which allows for updates to previously added sampling points.

As mentioned in Section 2.3, the ability to generate optimal sampling points will rely on the ability to invert the $\mathbf{A}$ matrix. As a result, we orthogonalize $\mathbf{A}$ before implementing the Greedy Algorithm in order to ensure the matrix does not become singular throughout the process. We construct $\mathbf{A}$ with an orthogonal basis using an Arnoldi-like iteration as described in [30]. In general, the Arnoldi-like iteration can be implemented using Algorithm 2 to generate the columns of an orthogonal matrix.

To implement the same Arnoldi-like iteration in 2D, we must first order the degrees of the polynomials. We order the polynomials by total degree starting with the highest degree in x and continuing towards the highest degree in y . The first few degrees of

Algorithm 2 Arnoldi-like Algorithm in 1D

```
q0 = b
q0 =  $\frac{b}{\|b\|}$ 
For  $k = 1 : N$ 
     $v = x * q_{k-1}$ 
    For  $j = 1 : k$ 
         $r = \langle q_j, v \rangle$ 
         $v = v - r * q_j$ 
    End
     $q_k = \frac{v}{\|v\|}$ 
End
```

the ordering look like

$$1 \rightarrow x \rightarrow y \rightarrow x^2 \rightarrow xy \rightarrow y^2 \rightarrow x^3 \rightarrow x^2y \rightarrow xy^2 \rightarrow y^3 \dots$$

Once this ordering is established, we implement the Arnoldi iteration to generate **A**. This process is detailed in Algorithm 3.

In the Matrix Orthogonalization Algorithm, k represents the total degree being calculated (the total degree is $k - 1$). Since, the q_0 column vector represents the zero degree vector, the k index starts at $k = 2$ (total degree 1). We generate subsequent column vectors by multiplying vectors from the previous total degree by either **x** or **y**. For example, the **y**³ vector will be gotten by multiplying the **y**² vector by **y**. Any column which includes at least one degree of **x** will be gotten by multiplying by **x**. For example, the **x**²**y**³ vector will be gotten by multiplying the **xy**³ vector by **x**.

Using the Matrix Orthogonalization Algorithm, we obtain a matrix which we use to calculate the Lebesgue function in the Greedy Algorithm. The matrices resulting from this process have low condition numbers, ensuring the invertibility of such matrices.

When implementing the Greedy Algorithm, we discretize the complex region and store the points as two vectors, $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$. These points represent the candidate points

Algorithm 3 Matrix Orthogonalization Algorithm

```
 $q_0 = \mathbf{1}$ 
 $q_0 = \frac{q_0}{\|q_0\|}$ 
count=0
For  $k = 2 : N$ 
  For  $j = 1 : k$ 
    count=count+1
    If  $j=k$ 
       $q_{\text{count}+1} = q_{\text{count}+1-k} * \hat{\mathbf{y}}$ 
    Else
       $q_{\text{count}+1} = q_{\text{count}+1-(k-1)} * \hat{\mathbf{x}}$ 
    End
    For  $p = 1 : \text{count}$ 
       $r(p) = q'_p * q_{\text{count}+1}$ 
       $q_{\text{count}+1} = q_{\text{count}+1} - r(p) * q(p)$ 
    End
     $q_{\text{count}+1} = \frac{q_{\text{count}+1}}{\|q_{\text{count}+1}\|}$ 
  End
End
```

from which the Greedy Algorithm selects from. Selected sampling points are then stored in separate vectors which we shall denote with $\mathbf{x}$ and $\mathbf{y}$. Since we will be adding the points from the discretization to the set of sampling nodes as we iterate throughout the greedy algorithm, it is important that we orthogonalize the Vandermonde matrix which includes the set of discretization points. We shall denote this orthogonalized matrix as $\hat{\mathbf{A}}$. We can then calculate $\mathbf{A}$, the Vandermonde matrix of only the selected sampling points by selecting the corresponding rows and columns of $\hat{\mathbf{A}}$ where the row corresponds the selected sampling point and the columns correspond to the orthogonalized basis columns.

To illustrate the effectiveness of the Matrix Orthogonalization Algorithm, let us consider the discretization of the unit circle depicted in Figure 3.1. This discretization contains 7668 candidate sampling points inside the unit circle. If we generate the

Vandermonde matrix, $\mathbf{A}$, where the rows corresponds to each of the 7668 candidate points and columns correspond to the first 400 Chebyshev basis functions in 2D (listed in Table 3.1), the matrix has a condition number of $\kappa(\mathbf{A}) = 1.13 \times 10^{10}$. If we apply the Matrix Orthogonalization Algorithm to the same points, we obtain $\kappa(\hat{\mathbf{A}}) = 1$, as expected.

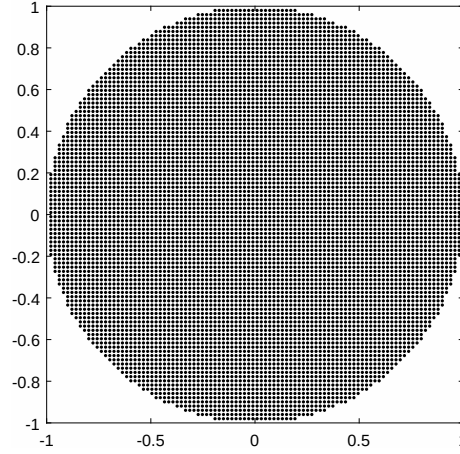


Figure 3.1: Unit Circle Discretization

$\phi_0(x, y)$	1
$\phi_1(x, y)$	$\cos(\arccos x)$
$\phi_2(x, y)$	$\cos(\arccos y)$
$\phi_3(x, y)$	$\cos(2 \arccos x)$
$\phi_4(x, y)$	$\cos(\arccos x) \cos(\arccos y)$
$\phi_5(x, y)$	$\cos(2 \arccos y)$

Table 3.1: Chebyshev Basis Functions in 2D

3.1.2 Numerical Experiments on Complex 2D Regions

We implement the algorithms on 2D regions. Some examples of the complex regions include the Peanut, Bumped-disk, Bumped-disk Ring, and L-shape region shown in Figure 3.2. In the 2D case, we start by sampling optimal nodes for the

least-squares problem rather than the interpolation problem to further ensure that the interpolation matrix will remain nonsingular. In this formulation, we include twice as many points as the degree of the interpolant. Thus, at each iteration of the greedy algorithm, we add two sampling points instead of one and the size of the resulting Vandermonde matrix increases by two rows and one column.

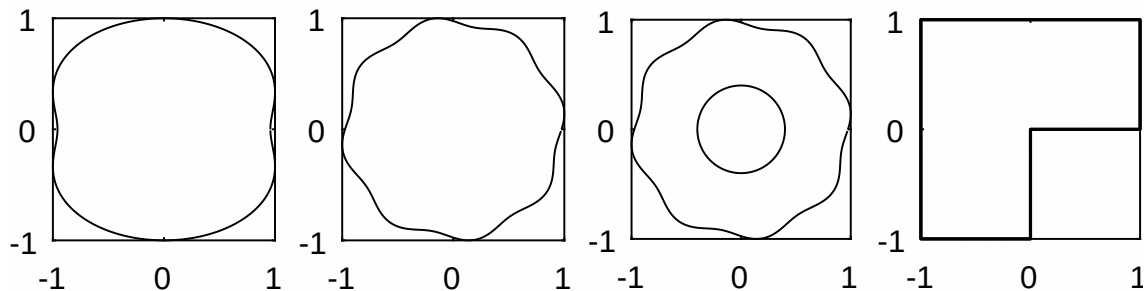


Figure 3.2: Complex 2D Regions: Peanut, Bumped-disk, Bumped-disk Ring, and L-Shape Region

For the Peanut region, we obtain 500 sampling points. Figure 3.3 plots the resulting sampling points as well as the behavior of the Lebesgue constant at each iteration. We see that the Lebesgue constant decreases rapidly in the first few iterations. This is due to the fact that the starting set of nodes was far from optimal. The Greedy Algorithm was able to use the sub-optimal starting set of nodes and improve it by adding subsequent sampling nodes. This led to a dramatic improvement in the Lebesgue constant. We see as we continue to add nodes, the Lebesgue constant grew overall in a slow steady rate. We note that there are various oscillations in the behavior of the Lebesgue constant. This can be attributed to the fact that we develop nested sets of nodes which are not able to change the location of previously placed nodes at any iteration.

The Bumped-disk Ring region presents similar results. Figure 3.4 plots the resulting sampling points as well as the behavior of the Lebesgue constant at each iteration. Again, we see that the Greedy Algorithm was able to take the sub-optimal starting set

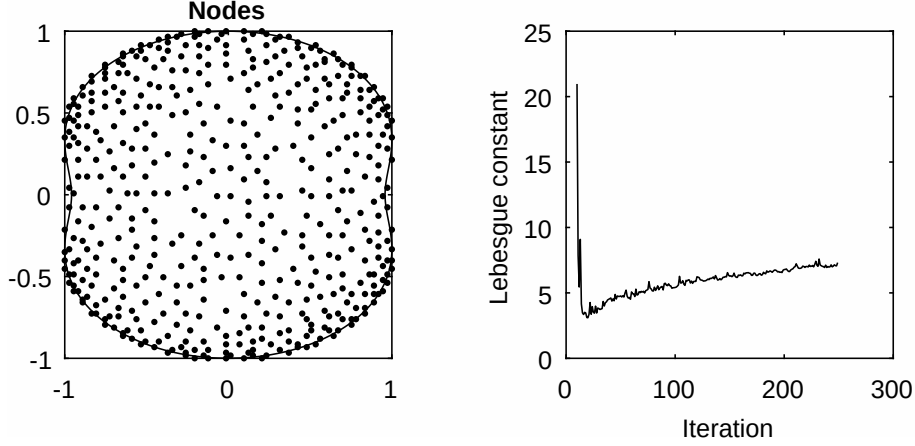


Figure 3.3: Peanut Region Greedy Algorithm Points. *Left:* The location of the 500 optimal sampling nodes found using the greedy algorithm. *Right:* The behavior of the Lebesgue constant per iteration of the greedy algorithm.

of nodes and improve it by adding more sampling nodes. This dramatically improves the Lebesgue Constant which then grows steadily as we add more and more nodes. Apparent in both the case of the Peanut and the Bumped-disk Ring is the clustering of the nodes. We see that sampling nodes tend to cluster towards the boundaries in both complex regions. This is expected as it is analogous to the 1D case where the clustering towards the edge of the domain mitigates instability and oscillatory behavior.

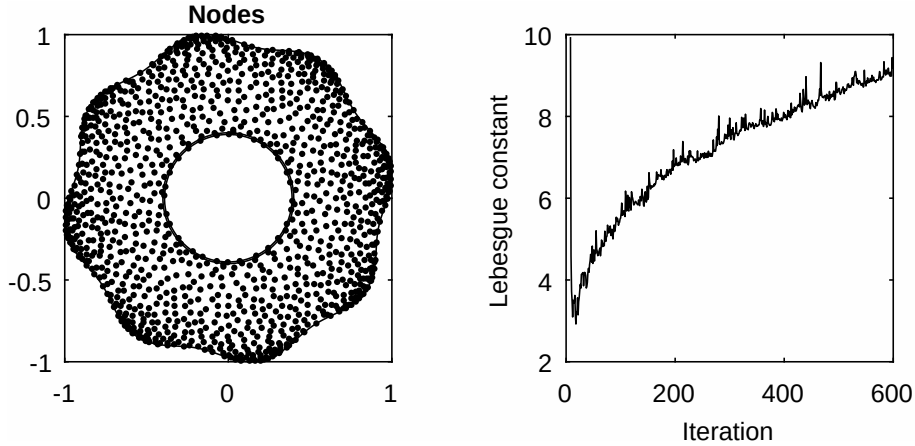


Figure 3.4: Bumped-disk Ring Region Greedy Algorithm Points. *Left:* The location of the 1200 optimal sampling nodes found using the greedy algorithm. *Right:* The behavior of the Lebesgue constant per iteration of the greedy algorithm.

Finally, we apply the greedy algorithm to the L-shape region. We obtain 10,000 points and observe similar behavior in the clustering and the growth of the Lebesgue constant as we saw in the Peanut and the Bumped-disk ring.

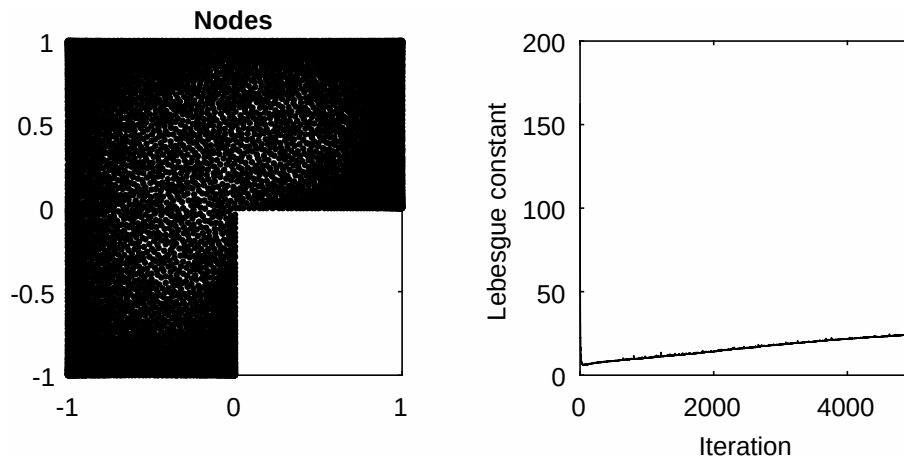


Figure 3.5: L-shape Region Greedy Algorithm Points. *Left:* The location of the 10,000 optimal sampling nodes found using the greedy algorithm. *Right:* The behavior of the Lebesgue constant per iteration of the greedy algorithm.

3.1.3 Saving Computation Time

Redefining Node Candidates Using a Density Function Approach

In the Greedy Algorithm approach, we add new sampling points at the location of the maximum of the Lebesgue function. Finding this maximum requires a discretization of the two-dimensional complex region in which we calculate the Lebesgue function at each point. Thus, if the task is to find a large number of sampling points, the resolution of the discretization must be fine enough in order to find the maximum of the Lebesgue function. Furthermore, the resolution of the discretization must be fine enough to ensure we can capture the appropriate clustering behavior that results from finding the optimal sampling points. Thus, as we increase the size of our sampling nodes, the resolution must increase as well. For a large number of sampling points, the computational cost on a discretization with the required resolution becomes a

limiting factor. This motivates the need to limit the number of node candidates required.

In order to limit the number of node candidates, we strategically place the node candidates so that computation time is not wasted calculating the Lebesgue function at poorly placed node candidates. To strategically place these nodes, we leverage the clustering behavior seen in the Peanut, Bumped-disk Ring, and L-shape region results. We notice that the optimal sampling nodes cluster near the boundaries of each complex region, and so, the general strategy will be to also concentrate the node candidates along the boundaries of the region.

We apply a density function approach in order to strategically place the nodes. We start by applying the Greedy Algorithm to find a relatively low number of sampling points. Since the number of sampling points desired here is low, we can start with an equispaced discretization. After iterating the Greedy Algorithm, we obtain an initial set of sampling nodes. From here, we generate a density function. This can be done by partitioning the complex region, and counting the number of sampling nodes in each partition. Then, we discretize the complex region again and place the node candidates according to the density function obtained by counting the sampling nodes from each partition. Doing so, we end up with a discretization with the node candidates clustered along the boundaries. This prevents the Greedy Algorithm from having to extraneously calculate the Lebesgue function values at a large number of points away from the boundary of the complex region. The overall process for obtaining optimal sampling points is summarized in Algorithm 4 below.

Algorithm 4 Optimal Sampling Points Using Density Function Discretization

1. Start with equispaced discretization of the complex region
 2. Implement the Matrix Orthogonalization Algorithm
 3. Implement Greedy Algorithm. Iterate to obtain a fraction of the desired number of sampling points
 - (a) Start with given set of interpolation points: $(x_j)_{j=0}^n$
 - (b) For the current set of interpolation points, calculate the Lebesgue function:
 $\lambda(x) = \sum_{j=0}^n |l_j(x)|$
 - (c) Find the maximum of the Lebesgue function
 - (d) Add new interpolation point at the location of the maximum: $x_{n+1} = \operatorname{argmax}_{[-1,1]} \lambda(x)$
 - (e) Iterate until the desired number of interpolation points is reached
 4. Generate new discretization of the complex region
 - (a) Partition the complex region
 - (b) Calculate the density function by counting the number of sampling points resulting from the Greedy Algorithm in Step 2 that belong to each partition
 - (c) Create discretization of the complex region by placing node candidates in each partition proportional to the density function value of the partition
 5. Implement the Matrix Orthogonalization Algorithm
 6. Implement Greedy Algorithm. Iterate to obtain the desired number of sampling points
 - (a) Start with given set of interpolation points: $(x_j)_{j=0}^n$
 - (b) For the current set of interpolation points, calculate the Lebesgue function:
 $\lambda(x) = \sum_{j=0}^n |l_j(x)|$
 - (c) Find the maximum of the Lebesgue function
 - (d) Add new interpolation point at the location of the maximum: $x_{n+1} = \operatorname{argmax}_{[-1,1]} \lambda(x)$
 - (e) Iterate until the desired number of interpolation points is reached
-

Leveraging Lagrange Polynomials in the Interpolation Case

We introduce another method to speed up computational time. This method applies to finding optimal sampling points for interpolation. We recall that in the Greedy Algorithm we need to calculate the Lebesgue function and locate the maximum. To do this, we evaluate the Lebesgue function, by summing the absolute value of the Lagrange polynomials. To find the Lagrange polynomials, we need to calculate the Lagrange polynomial coefficients obtained from $\mathbf{C} = \mathbf{A}^{-1}$. Then, $\hat{\mathbf{A}}\mathbf{C}$ gives us the matrix whose columns are the Lagrange polynomials evaluated at the (x, y) point represented in each row. Finally the Lebesgue function is obtained by taking the row sum of the absolute value of the elements in $\hat{\mathbf{A}}\mathbf{C}$.

We recall that with a discretization of the complex region stored in $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, we generate the Vandermonde matrix, $\hat{\mathbf{A}}$ using the Matrix Orthogonalization Algorithm. Further, at any iteration, we have the current set of sampling nodes stored in $\mathbf{x}$ and $\mathbf{y}$ which are subsets of $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, respectively.

To save limit computation time, we leverage the structure of the Lagrange polynomials. Since the Lagrange polynomials are defined as:

$$l_j(x_k, y_k) = \begin{cases} 1 & k = j \\ 0 & k \neq j \end{cases}$$

where $x_k \in \mathbf{x}$ and $y_k \in \mathbf{y}$, this is equivalent to

$$\hat{\mathbf{A}}\mathbf{C}_{k',j} = \begin{cases} 1 & k' = j \\ 0 & k' \neq j \end{cases}$$

where the row index k' is such that $\hat{\mathbf{x}}_{k'} = x_k$. Thus, for n sampling points, we have $(\hat{x}_{1'}, \hat{y}_{1'}) , \dots , (\hat{x}_{n'}, \hat{y}_{n'}) = (x_1, y_1) , \dots , (x_n, y_n)$

Now, we rearrange $\hat{\mathbf{x}}, \hat{\mathbf{y}}$, and $\hat{\mathbf{A}}$ so that the n sampling points are placed in the top n columns.

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_{1'} \\ \hat{x}_{2'} \\ \hat{x}_{3'} \\ \vdots \\ \hat{x}_{n'} \\ \hat{x}_{j_1} \\ \vdots \\ \hat{x}_{j_p} \end{bmatrix} \quad \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_{1'} \\ \hat{y}_{2'} \\ \hat{y}_{3'} \\ \vdots \\ \hat{y}_{n'} \\ \hat{y}_{j_1} \\ \vdots \\ \hat{y}_{j_p} \end{bmatrix} \quad \hat{\mathbf{A}} = \begin{bmatrix} \hat{a}_{1'1} & \hat{a}_{1'2} & \dots & \hat{a}_{1'n} \\ \hat{a}_{2'1} & \hat{a}_{2'2} & \dots & \hat{a}_{2'n} \\ \hat{a}_{3'1} & \hat{a}_{3'2} & \dots & \hat{a}_{3'n} \\ \vdots & \vdots & \dots & \vdots \\ \hat{a}_{n'1} & \hat{a}_{n'2} & \dots & \hat{a}_{n'n} \\ \hat{a}_{j_11} & \hat{a}_{j_12} & \dots & \hat{a}_{j_1n} \\ \vdots & \vdots & \dots & \vdots \\ \hat{a}_{j_p1} & \hat{a}_{j_p2} & \dots & \hat{a}_{j_pn} \end{bmatrix}$$

where $(\hat{x}_{j_i}, \hat{y}_{j_i})_{i=1}^p \in (\hat{\mathbf{x}}, \hat{\mathbf{y}}) \setminus (\mathbf{x}, \mathbf{y})$ and where p is the number of points in discretization that have not been chosen as a sampling point.

Then, we see that the matrix $\hat{\mathbf{A}}\mathbf{C}$ takes the form below. That is, by the definition of the Lagrange polynomials, reordering the points will give us the identity matrix at the top of the matrix that gives us the Lagrange polynomials.

$$\hat{\mathbf{A}}\mathbf{C} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 1 \\ p_{j_11} & p_{j_12} & \dots & p_{j_1n} \\ \vdots & \vdots & \dots & \vdots \\ p_{j_p1} & p_{j_p2} & \dots & p_{j_pn} \end{bmatrix}$$

Since we have only rearranged the rows of $\hat{\mathbf{A}}$, the Lebesgue constant we find by taking the maximum of the row sum of the absolute values of the elements in $\hat{\mathbf{A}}\mathbf{C}$ remains the same. At each iteration of the Greedy Algorithm, we add one sampling point. We again interchange the $n+1$ row of $\hat{\mathbf{x}}, \hat{\mathbf{y}}$, and $\hat{\mathbf{A}}$ with the row corresponding to the location of the maximum of the Lebesgue function. The Vandermonde matrix $\mathbf{A}$ grows by one column and one row since we are adding an interpolation point. Lastly, $\hat{\mathbf{A}}\mathbf{C}$ must also grow by one column. This column is added using the same Arnoldi iteration mentioned in the Matrix Orthogonalization Algorithm.

After adding the new sampling point, we must recalculate the Lebesgue polynomials in $\hat{\mathbf{A}}\mathbf{C}$. In the Greedy Algorithm, we computed the inverse of the Vandermonde matrix $\mathbf{A}$ at each iteration to obtain the Lebesgue polynomials. Instead of having calculate the matrix inversion, we can simply update $\hat{\mathbf{A}}\mathbf{C}$ with a matrix multiplication. To update $\hat{\mathbf{A}}\mathbf{C}$, we require the $n+1$ row of $\hat{\mathbf{A}}\mathbf{C}$ to also satisfy the Lagrange polynomial requirement. Thus, the following system needs to be satisfied:

$$\begin{bmatrix} 1 & 0 & \dots & 0 & b_1 \\ 0 & 1 & \dots & 0 & b_2 \\ 0 & 0 & \dots & 0 & b_3 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & b_n \\ \hat{a}_{(n+1)'1} & \hat{a}_{(n+1)'2} & \dots & \hat{a}_{(n+1)'n} & b_{n+1} \end{bmatrix} \mathbf{Q} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \quad (3.1)$$

The b_i 's result from the Arnoldi iteration. It is gotten by multiplying one of the previous columns of $\hat{\mathbf{A}}\mathbf{C}$, hence, only two of the b_i 's are nonzero. Now, the computational savings comes from not having to invert a matrix at each iteration. Instead, we calculate the LU factorization of the matrix on the left hand side in Equation 3.1. Since only one of the b_i 's above the diagonal is nonzero, $\mathbf{L}$ will be the identity matrix

except for the last row whose values will be

$$\begin{bmatrix} \hat{a}_{(n+1)'1} & \hat{a}_{(n+1)'2} & \dots & \hat{a}_{(n+1)'n} & 1 \end{bmatrix}.$$

Similarly, $\mathbf{U}$ is also the identity matrix except for two elements. Suppose r is the index such that $b_r \neq 0$. Then, the r th element of the last column of $\mathbf{U}$ will have value b_r and the last element in the last column of $\mathbf{U}$ will have value $b_{n+1} - b_r * a_{(n+1)'r}$. Thus, the LU factorization can be easily calculated along with its inverse. The matrices below depict the factorization assuming the nonzero b_i element is the n th element.

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 1 & 0 \\ \hat{a}_{(n+1)'1} & \hat{a}_{(n+1)'2} & \dots & \hat{a}_{(n+1)'n} & 1 \end{bmatrix}$$

$$\mathbf{L}^{-1} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 1 & 0 \\ -\hat{a}_{(n+1)'1} & -\hat{a}_{(n+1)'2} & \dots & -\hat{a}_{(n+1)'n} & 1 \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 1 & b_n \\ 0 & 0 & \dots & 0 & b_{n+1} - b_n * \hat{a}_{(n+1)'n} \end{bmatrix}$$

$$\mathbf{U}^{-1} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 1 & \frac{-b_n}{b_{n+1}-b_n*\hat{a}_{(n+1)'n}} \\ 0 & 0 & \dots & 0 & \frac{1}{b_{n+1}-b_n*\hat{a}_{(n+1)'n}} \end{bmatrix}$$

And so, we have explicit calculations for $\mathbf{L}$, $\mathbf{L}^{-1}$, $\mathbf{U}$, and $\mathbf{U}^{-1}$ which allowed us to quickly update the $\hat{\mathbf{A}}\mathbf{C}$ matrix without having to perform a matrix inversion at each iteration. Suppose at a given iteration we have n sampling points and N points in the discretized candidate set. This new matrix update has a computational cost of $O(2N + nN)$.

3.2 Optimal Points for Constrained Approximation

In the previous sections, we demonstrated how we can find near-optimal sampling points for approximation on complex 2D geometries by using a Greedy Algorithm. In this section, we discuss how to handle constrained approximation to solve PDEs on the same geometries. In particular, we discuss how one well-known point sampling algorithm can be modified for constrained approximation. This algorithm implements a QR factorization with column-pivoting on Vandermonde matrices to sample near-optimal points. As mentioned earlier, this method provides approximate-Fekete points which have been shown to be near-optimal for 2D geometries. We will demonstrate that the algorithm for finding approximate-Fekete points can be modified in order to handle accurate constrained approximation to solve PDEs on complex 2D geometries.

We consider PDEs of the following form:

$$\frac{\partial u}{\partial t} = \mathcal{F}(\Delta u, t, x, y), \quad (x, y) \in \Omega, \quad (3.2)$$

$$u(t, x, y) = 0, \quad (x, y) \in \partial\Omega, \quad (3.3)$$

$$u(0, x, y) = g(x, y), \quad (x, y) \in \Omega. \quad (3.4)$$

The goal will be to find sampling points with near-optimal Lebesgue constant that will allow us to solve the PDE given the constraint in Equation 3.3. Given a set of pairs, $\mathbf{x} = (x_j, y_j)_{j=0}^n \in \Omega$, with function values z_j we can approximate the Laplacian operator by taking the Laplacian of the approximation in Equation 2.19:

$$\Delta p(x, y) = \sum_{j=0}^n c_j \Delta \phi_j(x, y) \quad (3.5)$$

$$= \begin{bmatrix} \Delta \phi_0(x, y) & \dots & \Delta \phi_n(x, y) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad (3.6)$$

$$= \begin{bmatrix} \Delta \phi_0(x, y) & \dots & \Delta \phi_n(x, y) \end{bmatrix} V_n^{-1}[\mathbf{x}] \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix}. \quad (3.7)$$

Thus, to obtain the column vector of the Laplacian approximation at the collocation points, we have

$$\left[\Delta p(x_i, y_i) \right]_{i=0}^n = \mathbf{D}_2(\mathbf{x}) \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} \quad (3.8)$$

where $\mathbf{D}_2(\mathbf{x})$, also known as the differentiation matrix, is defined as:

$$\mathbf{D}_2(\mathbf{x}) = \begin{bmatrix} \Delta\phi_0(x_0, y_0) & \Delta\phi_1(x_0, y_0) & \dots & \Delta\phi_n(x_0, y_0) \\ \Delta\phi_0(x_1, y_1) & \Delta\phi_1(x_1, y_1) & \dots & \Delta\phi_n(x_1, y_1) \\ \vdots & \vdots & \ddots & \vdots \\ \Delta\phi_0(x_n, y_n) & \Delta\phi_1(x_n, y_n) & \dots & \Delta\phi_n(x_n, y_n) \end{bmatrix} V_n^{-1}[\mathbf{x}]. \quad (3.9)$$

3.2.1 Algorithmic Considerations

Again, the optimality of sampling points discussed is quantified with the Lebesgue constant. In general, sampling points gotten by finding the solution to the global optimization problem of minimizing the Lebesgue constant are deemed optimal; however, this solution is not always computationally feasible. For this reason, we must trade off optimality for computational feasibility, leading to what are called near-optimal points, in this case the approximate-Fekete points are provided instead of the optimal Fekete points. The column-pivoting QR algorithm provides a robust method to generate sampling points within computational limits that have Lebesgue constant close to the Lebesgue constant of the global minimum and is chosen because they accomplish this trade-off.

3.2.2 Sampling Near-optimal Points using Approximate-Fekete Points

We find near-optimal sampling points for constrained optimization using the well-known algorithm that produces approximate-Fekete points. Sommariva and Vianello [38] introduced an effective and efficient algorithm that calculates approximate-Fekete points on a given geometry. These sampling points only require QR factorizations as opposed to high dimensional global optimization solves. To understand the formulation of the Fekete points, we recall that the Lagrange polynomials can be calculated as the ratio of two Vandermonde determinants from Equation 2.7:

$$l_j(x, y) = \frac{\det(V_n[(x_0, y_0), (x_1, y_1), \dots, (x_{j-1}, y_{j-1}), (x, y), (x_{j+1}, y_{j+1}), \dots, (x_n, y_n)])}{\det(V_n[\mathbf{x}])}$$

where V_n is defined to be

$$V_n[\mathbf{x}] = V_n[(x_0, y_0), \dots, (x_n, y_n)] \quad (3.10)$$

$$= \begin{bmatrix} \phi_0(x_0, y_0) & \phi_1(x_0, y_0) & \phi_2(x_0, y_0) & \dots & \phi_n(x_0, y_0) \\ \phi_0(x_1, y_1) & \phi_1(x_1, y_1) & \phi_2(x_1, y_1) & \dots & \phi_n(x_1, y_1) \\ \phi_0(x_2, y_2) & \phi_1(x_2, y_2) & \phi_2(x_2, y_2) & \dots & \phi_n(x_2, y_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n, y_n) & \phi_1(x_n, y_n) & \phi_2(x_n, y_n) & \dots & \phi_n(x_n, y_n) \end{bmatrix}. \quad (3.11)$$

The Fekete points are good points for polynomial interpolation because they have low Lebesgue constant. The Fekete points are gotten by maximizing the Vandermonde determinant in the denominator of Equation 2.7. Interpolating with these points which satisfy the maximal Vandermonde determinant, we see that each Lagrange polynomial is bounded, $\|l_j\|_\infty \leq 1$. And so, the Lebesgue constant is also bounded, $\Lambda \leq n + 1$.

The process of calculating the Fekete points requires a high-dimensional global optimization problem, one which becomes computationally costly very quickly. In

order make the problem more computationally tractable, optimality is traded off and the approximate-Fekete points are produced. Instead of having to solve a global optimization problem, the approximate-Fekete points only require QR factorizations.

The process to generate the approximate-Fekete points includes another implementation of a greedy algorithm. This greedy algorithm is discussed in detail in [5, 7]. To find the approximate-Fekete points, we must first discretize the 2D geometry. The points within this discretization, $\mathbf{xx} = (x_i, y_i)_{i=1}^M \in \mathbf{G}$, are the candidate points from which we select the approximate-Fekete points. Then, supposing we aim to select N approximate-Fekete points, we first generate the Vandermonde matrix: $V_{N-1}[\mathbf{xx}] \in \mathbb{R}^{M \times N}$. The greedy algorithm is applied to the transpose of the Vandermonde matrix, $\mathbf{A} = V_{N-1}'[\mathbf{xx}]$ and is summarized below in Algorithm 5.

Algorithm 5 Greedy Volume Submatrix Algorithm

1. Select ind_1 as the index of the column of $\mathbf{A}$ with maximum length.
 2. Given indexes $ind_1, \dots, ind_k$, select ind_{k+1} such that the volume generated by columns $ind_1, \dots, ind_k, ind_{k+1}$ is maximal.
-

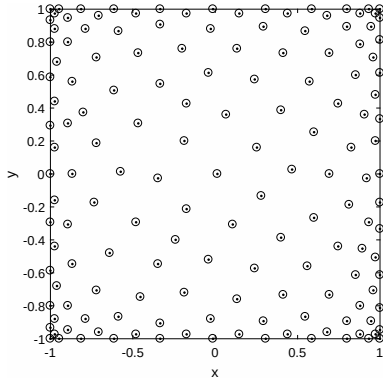
This algorithm greedily constructs the maximal volume submatrix of $\mathbf{A}$ and selects the points corresponding to the chosen columns to approximate the Fekete points. The algorithm can be easily implemented using QR factorizations with column-pivoting as detailed in [5]. In MATLAB notation, a 1D example of the Column-pivoting QR Algorithm is shown in Algorithm 6.

In Figure 3.6, we show the resulting approximate-Fekete points gotten by running the algorithm above. We obtain 153 sampling points for each of four 2D geometries. For the Square, the Padua points are known to have very low Lebesgue constant and are excellent approximation points on this region. In Figure 3.7, we plot 153 Padua points and note the similarities in the clustering of points between the Padua points

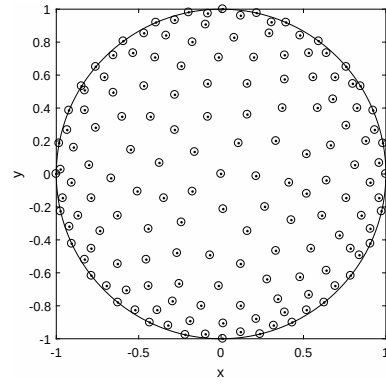
Algorithm 6 Example Column-Pivoting QR Algorithm

```
 $n = 21$ ; % number of interpolation points  
 $m = 1000$ ; % number of candidate points  
 $xx = linspace(-1, 1, m)$ ;  
 $A = gallery('chebvand', n, xx)$  % generate Vandermonde matrix with Chebyshev  
basis  
 $[Q, R, E] = qr(A, 'vector')$   
 $pts = xx(E(1:n))$ 
```

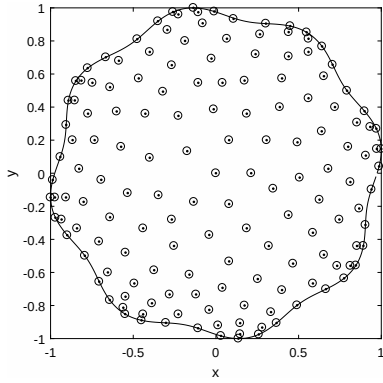
and the approximate-Fekete points.



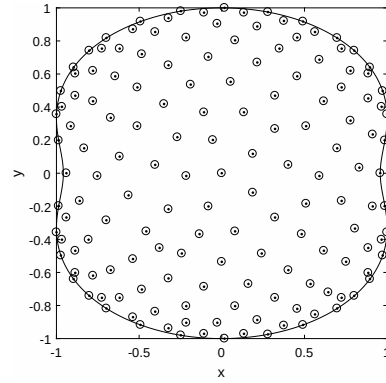
(a) Square Region Sampling Points



(b) Circle Region Sampling Points



(c) Bumped-disk Region Sampling Points



(d) Peanut Region Sampling Points

Figure 3.6: Approximate-Fekete Points

Furthermore, we list the Lebesgue constant of the resulting sampling point sets in Table 3.2 below. We see that the resulting Lebesgue constants for the approximate-Fekete points are not as optimal as the Lebesgue constant for the Padua points;

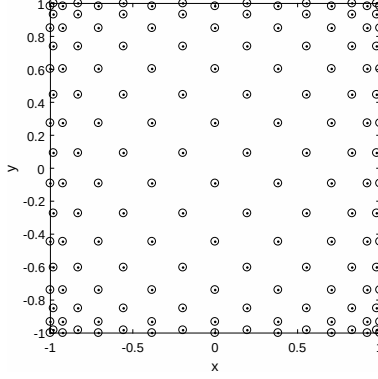


Figure 3.7: Square Region Padua Points

however, the Padua points are a special set of sampling points known only for the Square. The optimality of the sampling points in this case is traded off in order to obtain a robust algorithm capable of finding near-optimal sampling points on complex 2D regions.

Padua	Square	Circle	Bumped-disk	Peanut
8.41	27.22	31.70	29.26	38.58

Table 3.2: Lebesgue Constants for Approximate-Fekete Points and Padua Points

3.2.3 Comparing the Greedy Algorithm and the Column-Pivoting QR Algorithm

So far, we have considered two different methods for finding near-optimal sampling points. First, the Greedy Algorithm in Algorithm 1 and second, the Column-pivoting QR Algorithm in Algorithm 6. We will notice that each of these algorithms has their benefits and drawbacks.

The Greedy Algorithm was used in this chapter to find least-squares sampling points for complex 2D shapes by adding points, two at a time at the two maxima of the Lebesgue function. This algorithm is implemented to directly minimize the Lebesgue constant while remaining computationally feasible. Contrastingly, the Column-pivoting QR Algorithm does not directly minimize the Lebesgue constant.

We only know that the algorithm bounds the Lebesgue constant ($\Lambda \leq n + 1$).

Due to the nature of the Greedy Algorithm, the sets generated are nested. As a result, if we happen to add a poorly located sampling node, this cannot be undone in later iterations. Consequently, since the Column-pivoting QR Algorithm does not directly minimize the Lebesgue constant, this does not seem to be an issue for this algorithm.

Since the Column-pivoting QR Algorithm can be performed with just a few simple matrix factorizations in MATLAB, it finds sampling points quickly. Furthermore, the algorithm is quite robust; we simply need to have a set of candidate points and a basis for the Vandermonde matrix.

We compare the sampling points for the Peanut region from the Greedy Algorithm and the Column-pivoting QR Algorithm. In Figure 3.8, we plot the growth of the Lebesgue constant for the least squares problem for the two algorithms. The Lebesgue constants for the Greedy Algorithm are the same as the ones plotted in Figure 3.3. We will notice that the Column-pivoting QR Algorithm does not perform as well as the Greedy Algorithm in this case; however, the Lebesgue constants for these points are still low enough for accurate approximation. This is due to the fact that the approximate-Fekete points are not as well suited for the least squares problem as they are for the interpolation problem. To confirm this, we also compare the sampling points for the Peanut region from the two algorithms for the interpolation problem. These results are plotted in Figure 3.9. In this case, the Column-pivoting QR Algorithm performs better than the Greedy Algorithm.

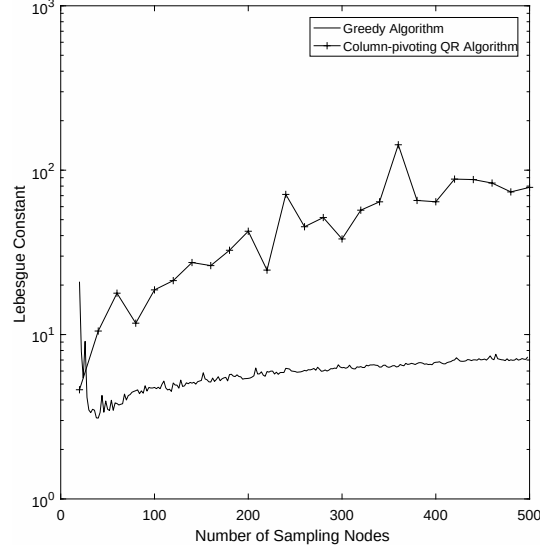


Figure 3.8: Algorithm Comparison for the Least Squares Problem

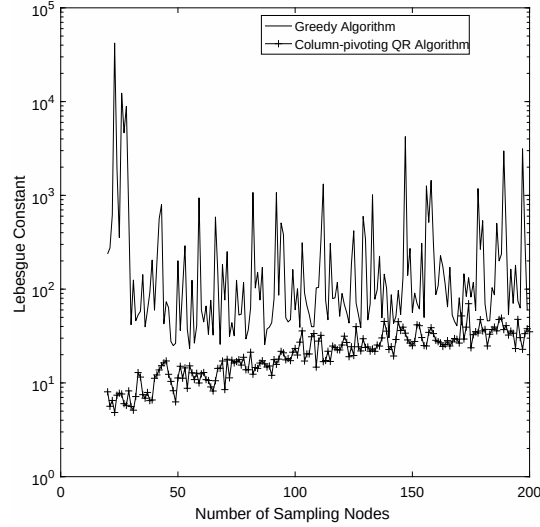


Figure 3.9: Algorithm Comparison for the Interpolation Problem

3.2.4 *Modifying the Column-Pivoting QR Algorithm for Constrained Approximation*

Having defined the Laplacian approximation in Section 3.2, the last step required is to modify the Column-Pivoting QR Algorithm to be able to handle the constrained approximation given in Equation 3.3. One way to modify the Column-pivoting QR Algorithm to handle the boundary conditions in Equation 3.3, requires us to first

choose a set of boundary points, $\mathbf{x}\mathbf{b} = (xb_j, yb_j)_{j=0}^m \in \delta\Omega$. Once a set of boundary points is obtained, the next step is to calculate the null space of the following matrix:

$$V_q[\mathbf{x}\mathbf{b}] = \begin{bmatrix} \phi_0(xb_0, yb_0) & \phi_1(xb_0, yb_0) & \dots & \phi_q(xb_0, yb_0) \\ \phi_0(xb_1, yb_1) & \phi_1(xb_1, yb_1) & \dots & \phi_q(xb_1, yb_1) \\ \phi_0(xb_2, yb_2) & \phi_1(xb_2, yb_2) & \dots & \phi_q(xb_2, yb_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(xb_m, yb_m) & \phi_1(xb_m, yb_m) & \dots & \phi_q(xb_m, yb_m) \end{bmatrix} \quad (3.12)$$

which we shall denote as $\mathbf{N}(q, \mathbf{x}\mathbf{b}) = \text{null}(V_q[\mathbf{x}\mathbf{b}])$.

This null space matrix will be used to implement a change of basis in order to select near-optimal sampling points for the constrained approximation. The Column-pivoting QR Algorithm is applied to the Vandermonde matrix, $V_{N-1}[\mathbf{x}\mathbf{x}] \in \mathbb{R}^{M \times N}$, where M is the number of candidate points in the discretization $\mathbf{G} \subset \Omega$ and N is the number of sampling points we wish to select. The algorithm for selecting near-optimal sampling points is modified so that instead of applying the Column-pivoting QR Algorithm to $V_{N-1}[\mathbf{x}\mathbf{x}]$, we apply it to $V_q[\mathbf{x}\mathbf{x}]\mathbf{N}(q, \mathbf{x}\mathbf{b})$. In order to obtain N sampling points, this matrix must contain N columns, thus, $q \geq N$ is chosen such that $\mathbf{N}(q, \mathbf{x}\mathbf{b})$ has N columns. Once q is chosen, the Vandermonde matrices, $V_q[\mathbf{x}\mathbf{x}]$ and $V_q[\mathbf{x}\mathbf{b}]$, are gotten by extending the degree, that is we add more basis columns until we have q columns.

With this change of basis, we ensure that the Vandermonde matrix evaluated at the boundary points satisfy $V_q[\mathbf{x}\mathbf{b}] = \mathbf{0}$, hence satisfying the constraint in Equation 3.3 while still allowing us to choose the sampling points using the Column-pivoting QR Algorithm. The modified algorithm can then be summarized in Algorithm 7 below.

Algorithm 7 Modified Column-Pivoting QR Algorithm

1. Discretize the boundary of the 2D geometry by selecting $\mathbf{xb} = (xb_j, yb_j)_{j=0}^m \in \delta\Omega$
 2. Given N , choose q such that $\mathbf{N}(q, \mathbf{xb}) = \text{null}(V_q[\mathbf{xb}])$ contains N columns.
 3. Obtain extended Vandermonde matrices, $V_q[\mathbf{xx}], V_q[\mathbf{xb}]$ by adding additional basis columns
 4. Apply a change of basis to the original Vandermonde Matrix by matrix multiplication: $V_q[\mathbf{xx}] \mathbf{N}(q, \mathbf{xb})$
 5. Implement the Column-pivoting QR Algorithm outlined in Section 3.2.2.
-

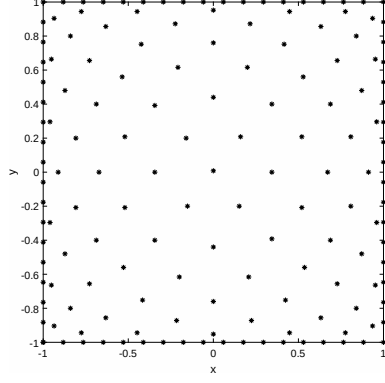
We note the presence of the Vandermonde matrix, $V_n[\mathbf{xx}]$, in Equation 3.9. Thus, if we wish to obtain approximations of the Laplacian to solve PDEs on complex 2D geometries, we could apply the modified algorithm. This can be done by simply substituting $V_q[\mathbf{xx}] \mathbf{N}(q, \mathbf{xb})$ for $V_n[\mathbf{xx}]$ in Equation 3.9.

3.2.5 Results

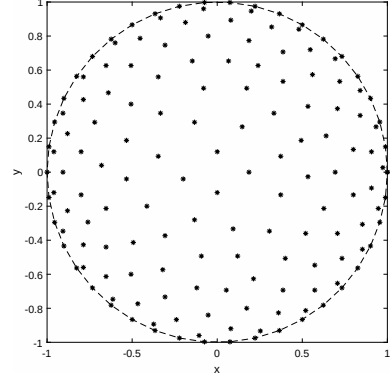
Applying the Modified Column-pivoting QR Algorithm, we can find optimal sampling points for complex 2D regions for constrained approximation. In Figure 3.10 below, we plot 153 optimal sampling points for constrained approximation for the four 2D geometries. These can be compared to the 153 approximate-Fekete points in Figure 3.6. Once we obtain the optimal sampling points, we wish to solve constrained PDEs to measure the accuracy of the resulting differentiation approximations.

Circle Region Results

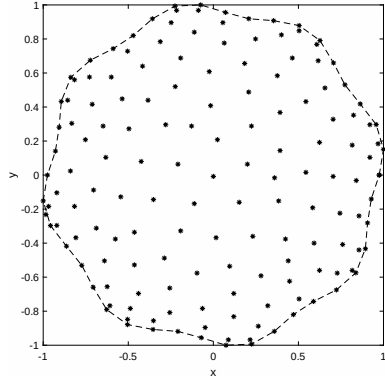
For the Circle region, we look to approximate the following PDE:



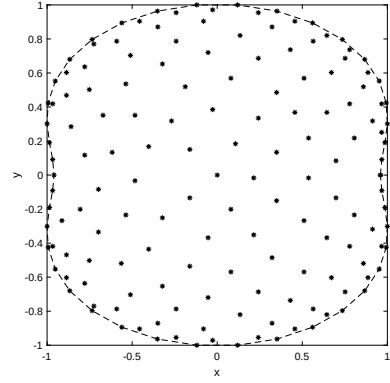
(a) Square Region Sampling Points



(b) Circle Sampling Points



(c) Bumped-disk Region Sampling Points



(d) Peanut Region Sampling Points

Figure 3.10: Optimal Sampling Points for Constrained Approximation

$$\begin{aligned} \frac{\partial u}{\partial t} &= \Delta u + \cos(t) \sin(1 - x^2 - y^2) \\ &\quad + 4 \sin(t) ((x^2 + y^2) \sin(1 - x^2 - y^2) + \cos(1 - x^2 - y^2)) \\ u_0 &= \mathbf{0} \end{aligned}$$

For this example, the solution is $u(t, x, y) = \sin(t) \sin(1 - x^2 - y^2)$. Applying a built-in MATLAB ODE solver to time-step using the Laplacian approximation resulting from the optimized sampling nodes for constrained approximation, we can measure the accuracy of the differential operator approximation. In Figure 3.11 below, we observe the error of the calculated solution at time $t = 10$ compared to the actual solution plotted against the square root on the number of sampling nodes used on

the interior of the Circle, n . We notice that for this example, the solution quickly converges.

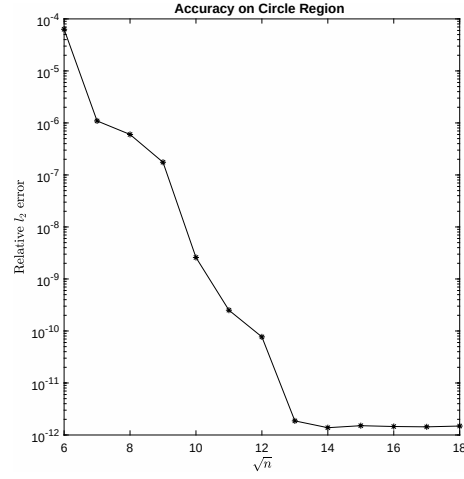


Figure 3.11: Solution Convergence on the Circle Region

Square Region Results

For the Square region, we look to approximate the following PDE:

$$\begin{aligned}\frac{\partial u}{\partial t} &= \Delta u + \sin(t) \sin(\pi x) \sin(\pi y) \\ u_0 &= \mathbf{0}\end{aligned}$$

In this case, we estimate the accuracy of solution at time $t = 10$ using node refinement. To do this, we obtain the solution using increasing number of sampling nodes and take the solution using the most nodes to be the true solution to which we calculate the error. We notice that for this example, the solution quickly converges in a manner similar to the Circle Region. In this case, we use 576 sampling nodes on the interior of the Square to obtain the solution we use as the true solution for node refinement. The convergence is shown in Figure 3.12 below.

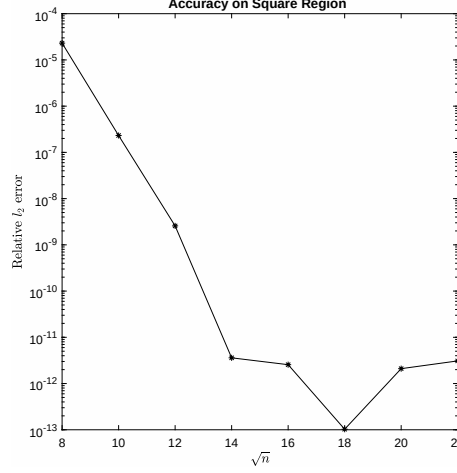


Figure 3.12: Solution Node Refinement Convergence on the Square Region

Peanut Region Results

For the Peanut region, we look to approximate the following PDE:

$$\frac{\partial u}{\partial t} = \Delta u + \sin(t)$$

$$u_0 = \mathbf{0}$$

Again, we estimate the accuracy of solution at time $t = 10$ using node refinement. In this case, we use 7095 sampling nodes on the interior of the Peanut to obtain the solution we use as the true solution for node refinement. The convergence is shown in Figure 3.13 below. We will notice here that the accuracy does not converge to as small of an error when compared to the Circle or Square Regions. We attribute this to the fact that the Peanut is a more complex region. As a result, we have to carry the approximation out to over 7000 points. We recall that we enforce the zero boundary condition in the Algorithm 7 by multiplying the Vandermonde matrix by the nullspace, $\mathbf{N}(q, \mathbf{x}\mathbf{b})$. Thus, this boundary condition enforcement is not exact, and for larger problems, roundoff errors may accumulate. As a result, we see in this example that the error does not converge to as low of an error as shown in the Circle

and Square regions.

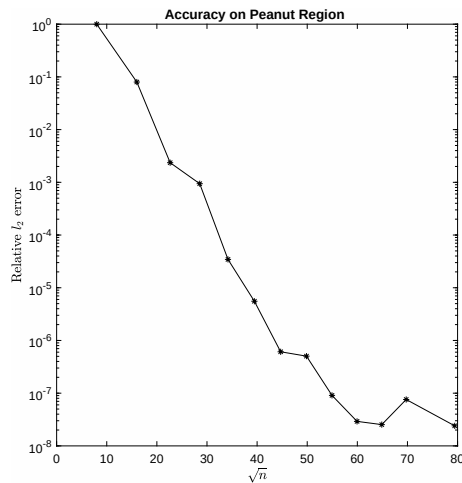


Figure 3.13: Solution Node Refinement Convergence on the Peanut Region

Bumped-disk Region Results

For the Bumped-disk region, we look to approximate the same PDE we used for the Peanut region:

$$\frac{\partial u}{\partial t} = \Delta u + \sin(t)$$

$$u_0 = \mathbf{0}$$

Again, we estimate the accuracy of solution at time $t = 10$ using node refinement. In this case, we use 6651 sampling nodes on the interior of the Bumped-disk to obtain the solution we use as the true solution for node refinement. The convergence is shown in Figure 3.14 below. We will notice that the accuracy behaves very similarly to the results of the Peanut region.

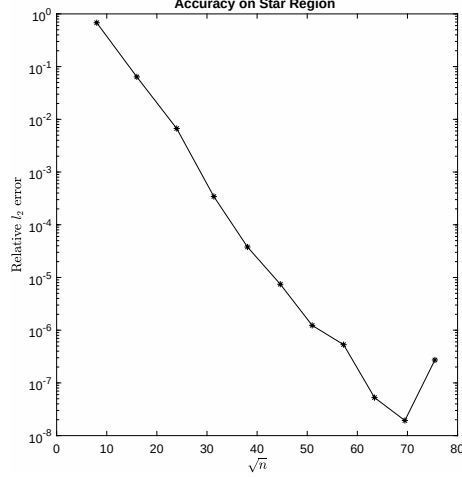


Figure 3.14: Solution Node Refinement Convergence on the Bumped-disk Region

3.3 Optimal Sampling for RBF Interpolation

So far, we have found points for data fitting using the polynomial basis. In this section, we focus on finding optimal sampling points using RBFs as the basis. RBFs present a good basis especially in higher dimensions when compared to polynomials. We notice that RBFs take distance metrics as inputs. Thus to change dimensions we simply need to be able to compute distances in the new dimension. No other change in basis is required when changing dimensions. This is not the case with the polynomial basis.

3.3.1 RBF Methods Background

A thorough introduction to RBFs methods can be found in [13, 16, 18]. We first formulate the RBF interpolation problem. Again, we assume a linear form. In this case, the RBF interpolant is a linear combination of translates of a radially-symmetric function denoted by $\phi(\|x - x_k\|)$. Examples of such functions can be found in [13, 16, 18]. Common examples of RBFs are listed in Table 3.3 and one such example is illustrated in Figure 3.15 below.

RBF	Basis Function	Parameter
Polyharmonic Spline	$\phi(r) = r^m$	$m \in 2\mathbb{N} - 1$
Multiquadric	$\phi(r) = \sqrt{1 + (\xi r)^2}$	$\xi \in \mathbb{R}$
Inverse Multiquadric	$\phi(r) = \frac{1}{1 + (\xi r)^2}$	$\xi \in \mathbb{R}$
Gaussian	$\phi(r) = e^{-(\xi r)^2}$	$\xi \in \mathbb{R}$

Table 3.3: RBF Examples

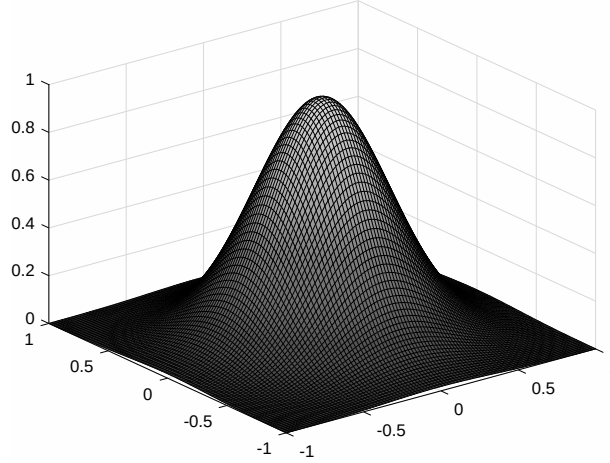


Figure 3.15: Gaussian RBF centered at the origin, $\xi = 2$

Interpolating through the points (x_j, y_j) , gives us the interpolant of the form

$$s(x) = \sum_{k=0}^n c_k \phi(\|x - x_k\|) \quad (3.13)$$

Again, we can formulate a linear interpolation system.

$$\begin{bmatrix} \phi(\|x_0 - x_0\|) & \phi(\|x_0 - x_1\|) & \dots & \phi(\|x_0 - x_n\|) \\ \phi(\|x_1 - x_0\|) & \phi(\|x_1 - x_1\|) & \dots & \phi(\|x_1 - x_n\|) \\ \phi(\|x_2 - x_0\|) & \phi(\|x_2 - x_1\|) & \dots & \phi(\|x_2 - x_n\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|x_n - x_0\|) & \phi(\|x_n - x_1\|) & \dots & \phi(\|x_n - x_n\|) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad (3.14)$$

This system is the same as the one mentioned in Equation 2.2 with the polynomial Vandermonde matrix being replaced by the RBF Vandermonde matrix. We note that

since the RBF Vandermonde matrix takes distance metrics as inputs, the form of our linear system in Equation 3.14 is independent of coordinate system and dimension. Therefore, the 2D setup keeps the same form as Equation 3.14 (using the l_2 norm for both the 1D and 2D cases).

If we want to find points for RBF interpolation using the Column-pivoting QR Algorithm, we would need to generate a Vandermonde matrix using the selected RBF basis, $\phi(r)$, and a set of candidate points. This Vandermonde matrix would look like:

$$\begin{bmatrix} \phi(\|x_0 - x'_0\|) & \phi(\|x_0 - x'_1\|) & \dots & \phi(\|x_0 - x'_n\|) \\ \phi(\|x_1 - x'_0\|) & \phi(\|x_1 - x'_1\|) & \dots & \phi(\|x_1 - x'_n\|) \\ \phi(\|x_2 - x'_0\|) & \phi(\|x_2 - x'_1\|) & \dots & \phi(\|x_2 - x'_n\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|x_m - x'_0\|) & \phi(\|x_m - x'_1\|) & \dots & \phi(\|x_m - x'_n\|) \end{bmatrix} \quad (3.15)$$

for $m + 1$ candidate points, $(x_0, \dots, x_m)$ and $n + 1$ centers, $(x'_0, \dots, x'_n)$. Thus, we see that in order for the RBF basis to be defined, we first have to know the location of the centers, a requirement which was not necessary for polynomials.

3.3.2 Finding Optimal Points for RBF Interpolation

Optimal nodes for RBF approximations have been studied in [11, 12, 31], to cite but a few. Most commonly, these methods attempt to minimize the power function associated with the approximate scheme. Similarly to the Lebesgue constant, the power function is the condition number of the approximation map for functions on the associated Native Space [11]. Functions in the Native Space; however, are known to be as smooth as the corresponding RBF kernel [42]. For this reason, we believe the power function underestimates the sensitivity of the method. In this section, we propose extending the Column-pivoting QR Algorithm, developed for polynomials, to

RBFs as we favor the use of the Lebesgue constant as the point conditioning measure.

In order to modify the Column-pivoting QR Algorithm for RBF interpolation, we follow a two-step process. We first select $n + 1$ centers, and then we select $n + 1$ interpolation points. To do this, we first generate the matrix:

$$A_{RBF} = \begin{bmatrix} \phi(0) & \phi(\|x_0 - x_1\|) & \dots & \phi(\|x_0 - x_m\|) \\ \phi(\|x_1 - x_0\|) & \phi(0) & \dots & \phi(\|x_1 - x_m\|) \\ \phi(\|x_2 - x_0\|) & \phi(\|x_2 - x_1\|) & \dots & \phi(\|x_2 - x_n\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|x_m - x_0\|) & \phi(\|x_m - x_1\|) & \dots & \phi(0) \end{bmatrix} \quad (3.16)$$

We then compute the column-pivoted QR factorization to obtain $AP = QR$. From here, centers are selected by choosing the points corresponding to the first $n + 1$ columns of AP . Now with the centers, $(x'_0, \dots, x'_n)$, selected, the Vandermonde matrix in Equation 3.15 can be populated. Then, the same Column-pivoting QR Algorithm can be again applied to find the $n + 1$ desired optimal sampling points.

3.3.3 Results in 2D

Using the method to find optimal points for RBF interpolation discussed in Section 3.3.2, we find optimal points for the Peanut and the Bumped-disk Ring regions. For the Peanut region, we found 492 optimal sampling points as depicted in Figure 3.16. This was done using Gaussian RBFs with $\xi = 3.7$. The Lebesgue constant for these points is $\Lambda = 32.99$. For comparison, if we took equispaced points for the Peanut region, shown in Figure 3.17, we would have a Lebesgue constant of $\Lambda = 3.55 \times 10^4$. We plot the Lebesgue functions for these points in Figure 3.18 below.

Similarly, for the Bumped-disk Ring region, we found 492 optimal sampling points as depicted in Figure 3.19. This was done using Gaussian RBFs with $\xi = 3.9$.

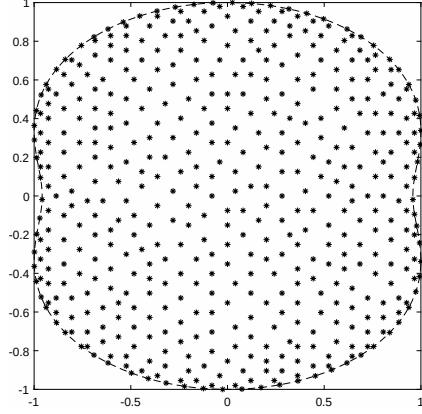


Figure 3.16: Optimal Samplings Points for RBF Interpolation on the Peanut

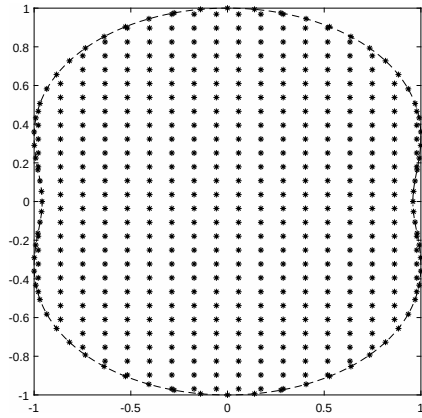
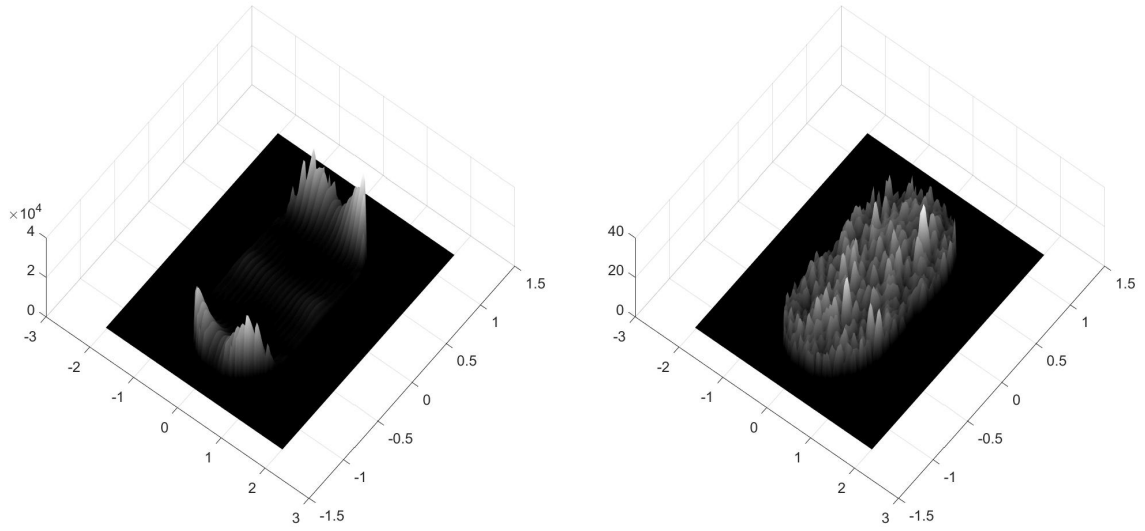


Figure 3.17: Equispaced Points for RBF Interpolation on the Peanut

The Lebesgue constant for these points is $\Lambda = 22.12$. For comparison, if we took equispaced points for the Bumped-disk Ring region, shown in Figure 3.20, we would have a Lebesgue constant of $\Lambda = 692.92$. We plot the Lebesgue functions for these points in Figure 3.21 below.

3.3.4 Results for Surfaces and 3D Regions

We can also find optimal sampling points for RBF approximations in 3D regions. We note that since the RBF basis does not depend on the dimension of the space, translating the algorithms into 3D does not require any modification. The approximation setup is the same as the setup for 2D, thus we can apply the same



(a) Lebesgue Function for Equispaced Peanut Points (b) Lebesgue Function for Optimal Peanut Points

Figure 3.18: Lebesgue Function for Peanut Region Points

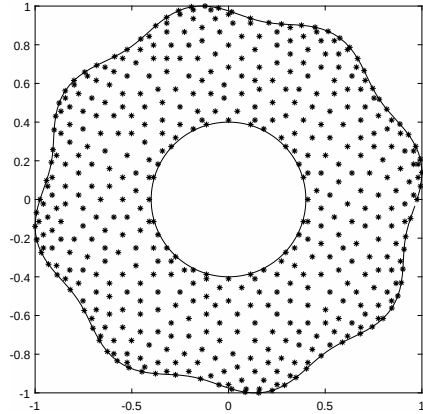


Figure 3.19: Optimal Samplings Points for RBF Interpolation on the Bumped-disk

Column-pivoting QR Algorithm to find both centers and sampling points. In 3D, we first consider the surfaces of the Half-Torus and the Hyperboloid.

For the Half-Torus surface, we compare the Lebesgue function of the optimal sampling points to the Lebesgue function of the angularly equispaced points. The points as well as the Lebesgue functions are shown in Figure 3.22 below. We notice a

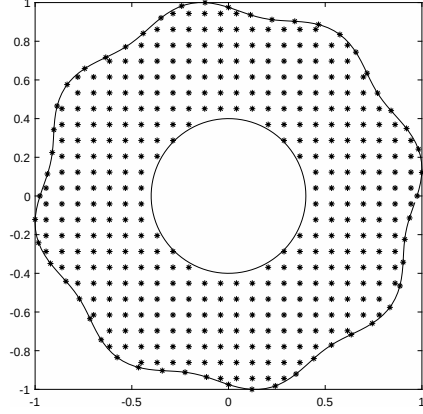
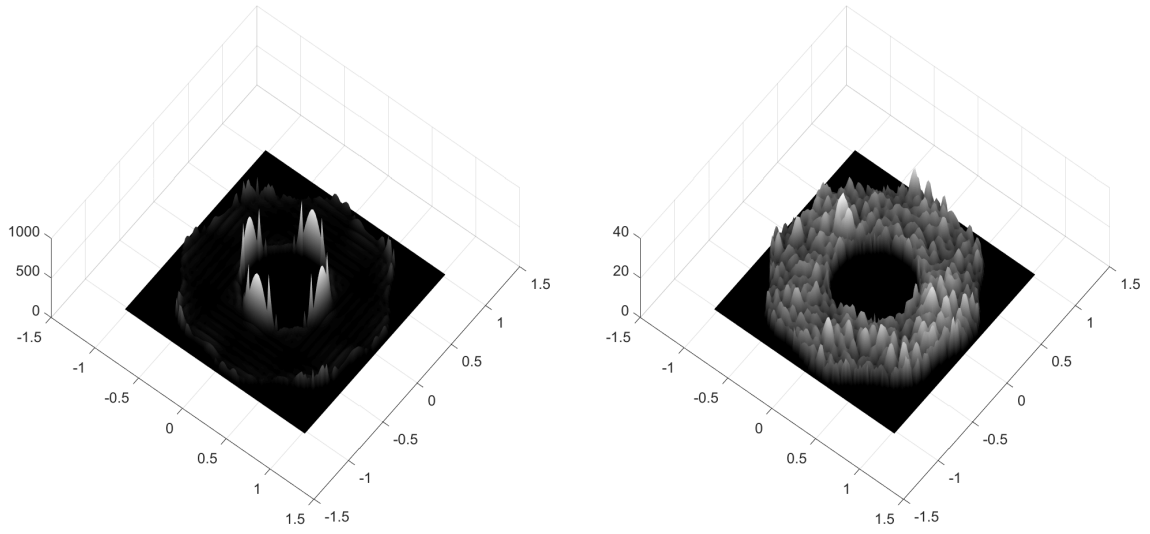


Figure 3.20: Equispaced Points for RBF Interpolation on the Bumped-disk

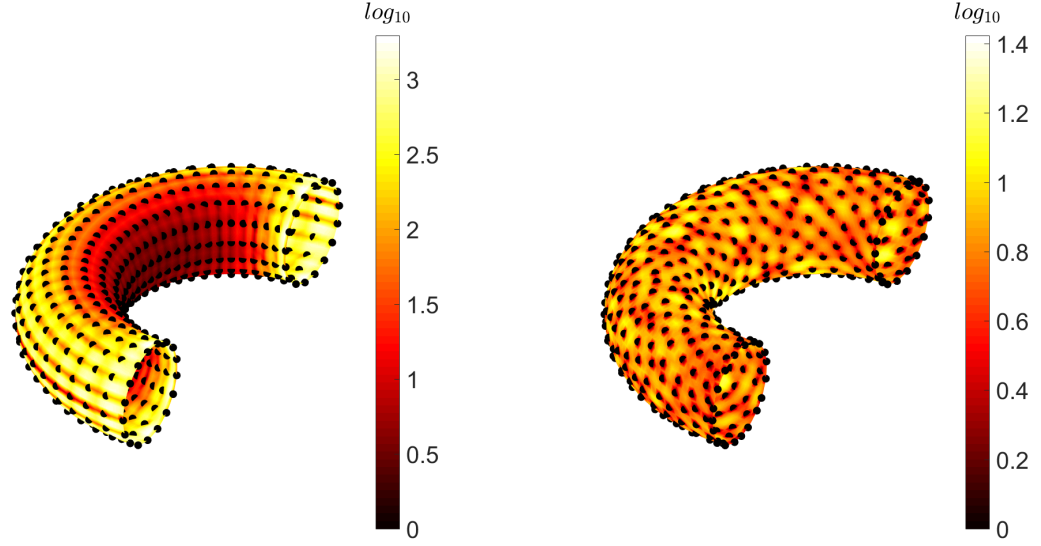


(a) Lebesgue Function for Equispaced Bumped-disk Points (b) Lebesgue Function for Optimal Bumped-disk Points

Figure 3.21: Lebesgue Function for Bumped-disk Region Points

significant difference in Lebesgue constant. For the angularly equispaced points, the Lebesgue constant is $\Lambda = 1.97 \times 10^3$ while the Lebesgue constant for the optimized points is $\Lambda = 26.50$.

The points as well as the Lebesgue functions for the Hyperboloid surface are shown in Figure 3.23 below. We again see a significant difference in Lebesgue constant. For

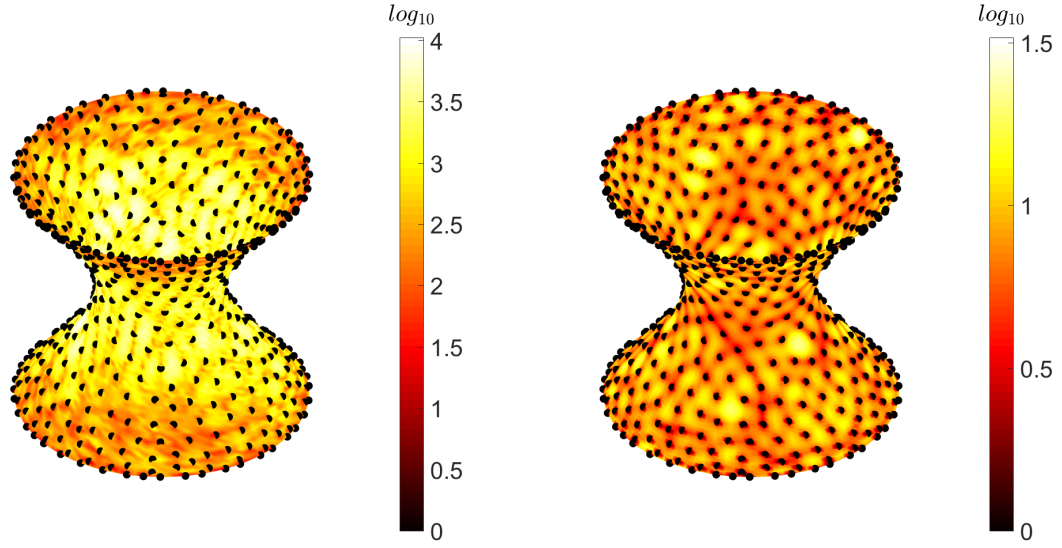


(a) Lebesgue Function for Angularly Equi- (b) Lebesgue Function for Optimal Half-Torus
paced Half-Torus Surface Points Surface Points

Figure 3.22: Lebesgue Function for Half-Torus Surface Points

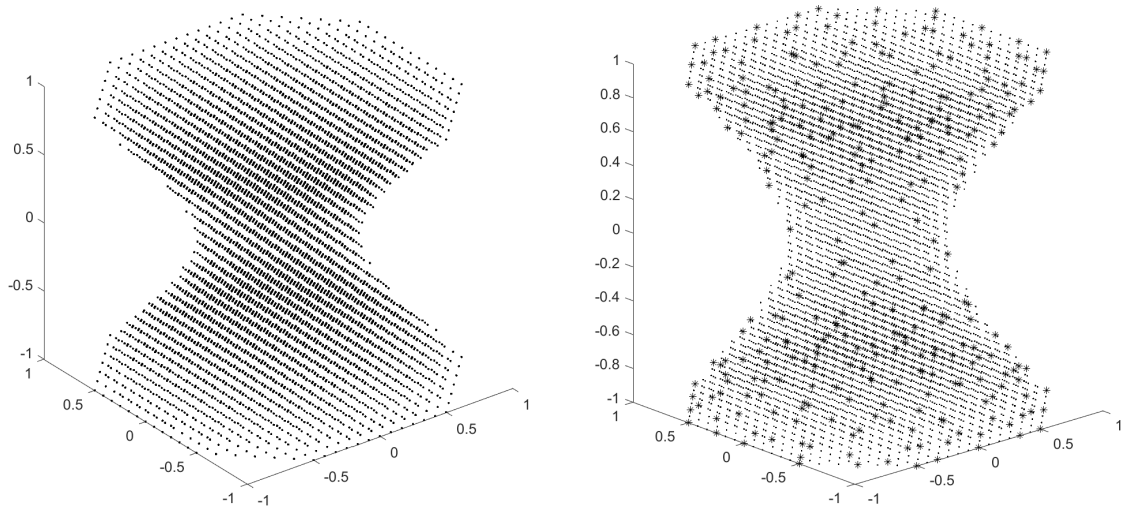
the angularly equispaced points, the Lebesgue constant is $\Lambda = 1.06 \times 10^4$ while the Lebesgue constant for the optimized points is $\Lambda = 32.83$.

We can also consider the entire volume of 3D geometries. For example, Figure 3.24 below depicts the candidate points for the Hyperboloid as well as 300 optimal points we find. The Lebesgue constant for the selected 3D optimal points is $\Lambda = 22.68$.



(a) Lebesgue Function for Angularly Equi- (b) Lebesgue Function for Optimal Hyper-
paced Hyperboloid Surface Points boloid Surface Points

Figure 3.23: Lebesgue Function for Hyperboloid Surface Points



(a) Hyperboloid Candidate Points (b) Optimal Hyperboloid Points and Candi-
date Points

Figure 3.24: Hyperboloid 3D Sampling Points

Chapter 4

SAMPLING NODES FOR HIGH-ORDER FINITE DIFFERENCE METHODS

We now shift our focus to finding optimal sampling points for finite difference methods in both 1D and 2D. In Chapter 3, we found sampling points which gave us optimal approximations using polynomial and RBF expansions. One way to investigate the behavior of derivatives is through spectral differentiation methods. That is, we formulate our interpolant and approximate derivatives by differentiating the interpolant, multiplying function values by a differentiation matrix, as discussed in Section 3.2. These methods provide high accuracy derivative approximations; however, they are costly to implement. Finite difference methods are used in order to reduce the computational cost of obtaining differentiation approximations. To approximate derivatives at a point, x_i , finite difference methods use function values from a set of neighboring sampling nodes, while spectral methods take into account all sampling nodes. As a result, differentiation matrices resulting from finite difference methods are sparse, while those resulting from spectral methods are not.

4.1 Polynomial Finite Difference Methods

4.1.1 *Background*

Recall the linear system of the interpolation setup. Equations 4.1 - 4.6 show how one can derive the 1D second order differentiation matrix based on an approximation

scheme.

$$\begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \dots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \phi_2(x_n) & \dots & \phi_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\mathbf{A}\mathbf{c} = \mathbf{y} \quad (4.1)$$

$$p(x) = \sum_{j=0}^n c_j \phi_j(x) \quad (4.2)$$

$$p(x)'' = \sum_{j=0}^n c_j \phi_j''(x) \quad (4.3)$$

$$= \begin{bmatrix} \phi_0''(x) & \phi_1''(x) & \dots & \phi_n''(x) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad (4.4)$$

$$= \begin{bmatrix} \phi_0''(x) & \phi_1''(x) & \dots & \phi_n''(x) \end{bmatrix} \mathbf{A}^{-1} \mathbf{y} \quad (4.5)$$

$$= \mathbf{D}_2 \mathbf{y} \quad (4.6)$$

We see that $\mathbf{D}_2 \mathbf{y}$ depends on the location of the nodes and the choice of basis. The goal is to find the optimal placement of the sampling nodes for high-order finite difference methods. Given an integer k , we wish to find the optimal placement of sampling nodes for finite difference methods that take the k nearest neighbors into derivative approximations. In this work k is taken to be odd to enforce a centered finite difference structure. We find optimal sampling points by investigating the behavior of the Lebesgue constant. Stability regions of the resulting differentiation

matrix operators are also taken into consideration.

Previous works which have found sampling points for finite difference methods include [21, 22, 23]. References [21, 22] construct stable high-order central difference methods by adding a small number of additional sampling points near the boundary. The central difference methods are stabilized for order up to 22 by the addition of three nodes near the boundary. In [23], optimal sampling points are obtained by minimizing the error formula found in Theorem 2.8 using a piecewise polynomial setup.

Since finite difference methods only consider nearby sampling points, we must implement interpolant approximations which behave in the same manner. Piecewise interpolation provides these desired traits.

4.1.2 Piecewise Polynomial Formulation

We first formulate finite difference methods based on piecewise polynomial interpolants. In 1D on the interval $[-1, 1]$, we develop individual interpolants based on the stencil size required. For example, with $m = 4$ sampling nodes, (x_0, x_1, x_2, x_3) , we may choose to use a stencil size of $k = 3$. In 1D, there will be $m - k + 1$ piecewise interpolants. In this case, the 2 stencils will have the groupings circled in Figure 4.1.

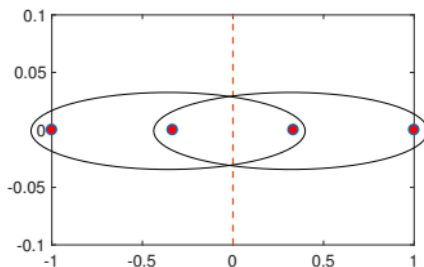


Figure 4.1: Piecewise Polynomial Stencil Grouping for $m = 4, k = 3$

The linear interpolation system for a given piecewise polynomial interpolant remains the same as Equation 2.2; however, the Vandermonde matrix will only include

entries for the 3 sampling nodes used in the stencil for the given piecewise polynomial interpolant.

We now consider the Lagrange polynomials. To find the coefficients for the Lagrange polynomials, we must invert the Vandermonde matrix; however, the Vandermonde matrix will be different for each stencil grouping. To evaluate the Lagrange polynomial at a point x^* , we must find its k -nearest neighbors in order to determine which stencil to use. From Figure 4.1, it is clear that if $x^* \leq 0$, the stencil (x_0, x_1, x_2) will be used while if $x^* > 0$, the stencil (x_1, x_2, x_3) will be used. For $x^* = 0$, a tiebreaker rule is implemented. Thus, the 4 Lagrange polynomials must be defined in a piecewise manner on the intervals $[-1, 0]$ and $(0, 1]$, using the Vandermonde matrix based on the stencil used for each interval. Figure 4.2 plots each of the resulting Lagrange polynomials.

The first Lagrange polynomial is a quadratic on the interval $[-1, 0]$ with value 1 at x_0 and 0 at $\{x_1, x_2\}$. The polynomial takes on value 0 on the rest of the interval since on this interval, x_0 is not used in the calculation of the polynomial. The second Lagrange polynomial is a quadratic on the interval $[-1, 0]$ with value 1 at x_1 and 0 at $\{x_0, x_2\}$. On the interval $(0, 1]$, it is a quadratic with value 1 at x_1 and 0 at $\{x_2, x_3\}$. The last two Lagrange polynomials are symmetric reflections of the first two.

Once we obtain each of the Lagrange polynomials, we derive the Lebesgue function by summing the absolute values of each Lagrange polynomial. This example's Lebesgue function is shown in Figure 4.3

As mentioned above, previous work on finding optimal sampling points for high-order finite difference methods using piecewise polynomials can be found in [23]. Sampling points in [23] are found by minimizing the error formula found in Theorem 2.8 for the Lagrange interpolation. In the piecewise setting, this formula is defined

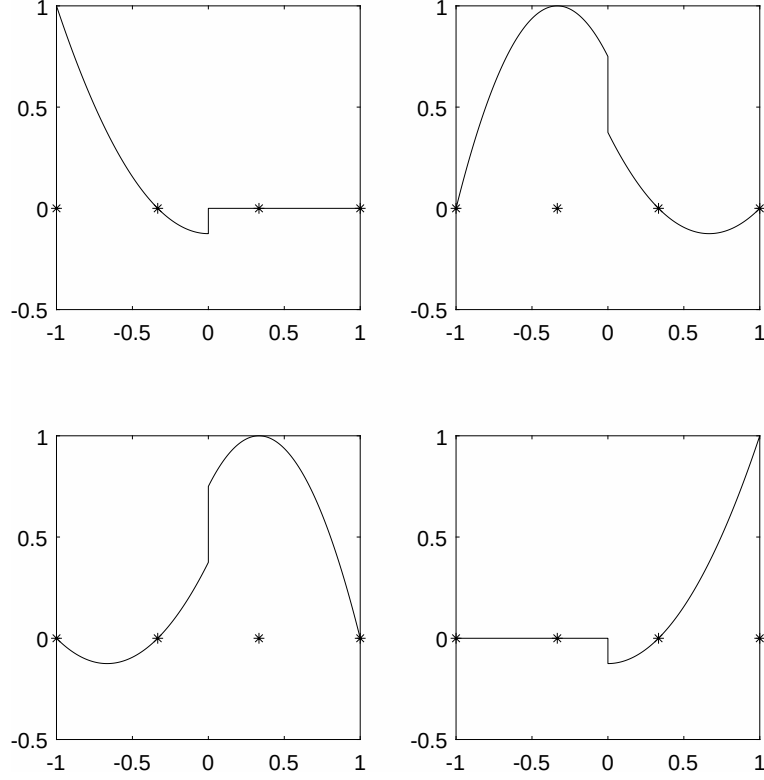


Figure 4.2: Lagrange Piecewise Polynomials. *Clockwise starting from the top left:* Lagrange Polynomial 1, Lagrange Polynomial 2, Lagrange Polynomial 4, Lagrange Polynomial 3.

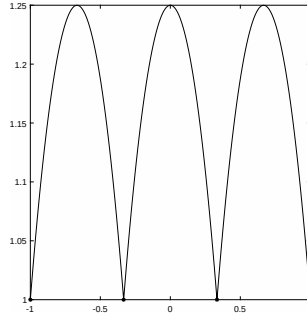


Figure 4.3: Lebesgue Function for Piecewise Polynomial Example

as:

$$\epsilon_i(x) = u(x, t) - I_i(x) \quad (4.7)$$

$$= \pi_i(x) \frac{u^{(k)}(\zeta)}{(k)!}, \quad (4.8)$$

where i is the stencil index, I is the interpolant, ζ is in the smallest interval that

contains the i th stencil, and $\pi_i(x)$ is the polynomial defined as:

$$\pi_i(x) = \prod_{j \in S_i} (x - x_j) \quad (4.9)$$

where S_i contains the indexes of the sampling points included in the i -th stencil. Consequently, the optimal sampling points for finite difference methods in [23], are found by minimizing the magnitude of the $\pi_i(x)$ terms.

The goal is to find the optimal placement of m sampling nodes using the Lebesgue constant as the objective function. For large m the dimension of this optimization problem becomes problematic. The same curse of dimensionality appeared in finding optimal sampling nodes for polynomial approximation methods. In Chapter 3, we implemented greedy algorithms to make the optimization problem more tractable. As a result, optimality was traded off in exchange for computational requirements. The same must be done to find optimal sampling points for finite difference methods. The goal is to reduce the dimension of the optimization problem. We introduce two dimension reduction methods.

4.1.3 Partial Centered Finite Difference Dimension Reduction Method

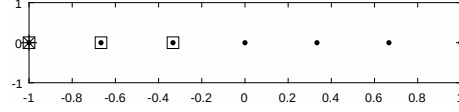
In the first method we impose a structure on a subset of the m sampling points. In the case where we have $m \geq k + 2$ sampling nodes, we can use centered finite difference for the nodes far enough away from the boundary, similar to the methods mentioned in [23]. An example of this method is illustrated in Figure 4.4 ($m = 7$, $k = 3$). In this example, the differentiation point in stencils 2-6 are centered. In stencils 1 and 7, the differentiation stencils are off-center due to their proximity to the boundary. In general, the first $\frac{k-1}{2}$ stencils and the last $\frac{k-1}{2}$ stencils are not centered, while those in between are. To find the optimal placement of all the sampling nodes, we apply a centered finite difference structure to the $m - 2k$ nodes in the interior

of the interval. The problem that remains is to find the optimal placement of the remaining $2k$ sampling nodes. Furthermore, we can impose symmetry at the origin. Thus, we only need to find the placement of the k sampling nodes on the boundary near $x = -1$ and reflect the points about $x = 0$. As a result, the dimension of the optimization problem is reduced from m to k . To find the solution, we implement a global minimum finder such as MATLAB's MultiStart [1].

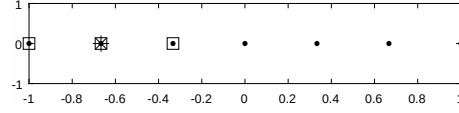
One important result of the formulation of this method is that increasing the number of sampling points does not dramatically increase the computational complexity of this optimization problem. This is due to the fact that the interior nodes remain stationary based on their centered finite difference configuration. The factor that does directly affect the computational requirements of this method is the stencil size. If we increase the stencil size (and the order of the finite difference method), we increase the dimension of the optimization problem. This will lead to massive increases in the computational requirements as dictated by the curse of dimensionality. Section 4.1.4 introduces a dimension reduction method where the dimension of the optimization space is independent of the stencil size.

A global minimum finder is used to find the optimal placement of the k boundary nodes. The function that is inputted calculates the Lebesgue constant of the given set of nodes. The minima finder used is MATLAB's *fmincon*. Constraints are required to keep the sampling nodes within the $[-1, 1]$ interval. Further, constraints are used to ensure the interior sampling nodes, which use centered finite difference points, are not moved throughout the optimization process. Lastly, we also use MATLAB's *MultiStart* in order to start multiple initial guesses. This improves our chances of finding the true global minimum, rather than local minima.

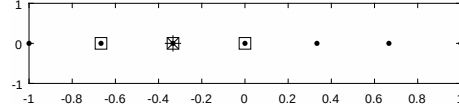
In the first example, we implement the global minimum finder to find optimal sampling points for the case with 50 sampling points, using a 9 point stencil. We



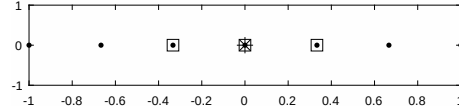
(a) Stencil 1



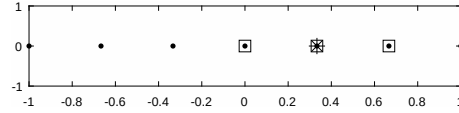
(b) Stencil 2



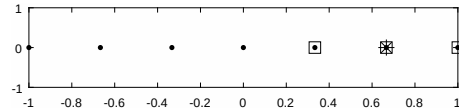
(c) Stencil 3



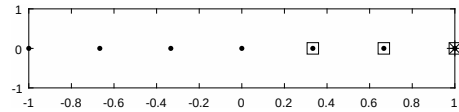
(d) Stencil 4



(e) Stencil 5



(f) Stencil 6



(g) Stencil 7

Figure 4.4: Differentiation Stencils. The red asterisk marks the differentiation point. The boxed points marks which points are used in the finite difference calculation. Stencils 2-6 use a centered finite difference structure while the boundary stencils, stencils 1 and 7 do not.

create a function that calculates the Lagrange piecewise polynomials and the Lebesgue function and outputs the Lebesgue constant for a given set of nodes, as outlined in Section 4.1.2. This output is then passed into *fmincon* and *MultiStart* in order to find

the set of sampling points with the lowest Lebesgue constant. We plot the results in Figure 4.5.

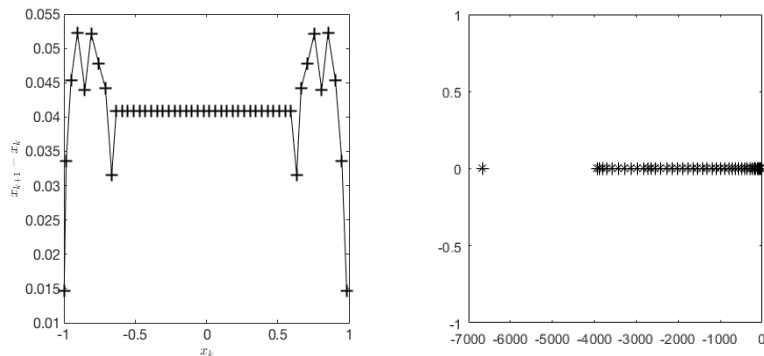


Figure 4.5: Partial Centered Finite Difference Method Example 1. *Left:* Spacing of optimal sampling points for the example with 50 sampling points and a 9 point stencil. *Right:* The eigenvalues of the resulting differentiation matrix in the real/imaginary plane.

We notice that the spacing of the optimal sampling points are very similar to those resulting from minimizing the error formula (Equation 4.9) in [23]. The resulting Lebesgue constant in this example is $\Lambda = 1.8538$. After obtaining the optimal sampling points, we calculate the differentiation matrix from Equation 4.16. In order to solve PDEs, we wish to be able to time-step in time. To do this in a stable manner, we would require that the eigenvalues for spatial differentiation matrices to have negative real parts. We observe that all eigenvalues of the differentiation matrix lie on the negative real axis. Thus, in this example, minimizing the Lebesgue constant leads to a stable finite difference method.

In the next example, we consider using a 15 point stencil. In this case, we apply this stencil to the case with 1000 sampling points in total. Figure 4.6 plots the results for this example. We notice the similarity in the spacing of the boundary points with the first example. The minimized Lebesgue constant is $\Lambda = 2.2314$. Again, we observe that the node placement with minimized Lebesgue constant leads to a stable method.

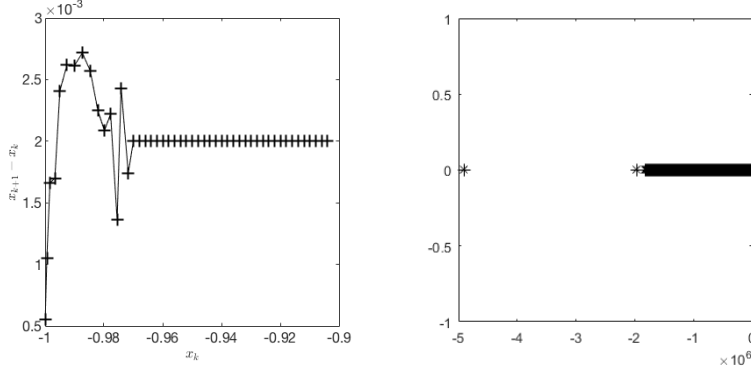


Figure 4.6: Partial Centered Finite Difference Method Example 2. *Left:* Spacing of optimal sampling points for the example with 1000 sampling points and a 15 point stencil. The spacing for the first 50 points is plotted. The spacing for the last 50 points is symmetric to the first 50 with the rest of the points in the middle having even spacing. *Right:* The eigenvalues of the resulting differentiation matrix in the real/imaginary plane.

4.1.4 KTE Dimension Reduction Method

In the Partial Centered Finite Difference method, we fixed the interior sampling points by imposing a centered finite difference structure and optimized the placement of the sampling points near the boundary. This led us to be able to fix the dimension of the optimization space. In the next method, we impose a different structure to the sampling points. This structure is gotten using the Kosloff and Tal-Ezer (KTE) mapping. This mapping was first introduced in [25] for Lebesgue constant applications. The KTE mapping is defined to be

$$x_j^{kte(\alpha)} := \frac{\arcsin(\alpha x_j^{cheb})}{\arcsin(\alpha)}, \quad j = 1, \dots, m. \quad (4.10)$$

where $x_j^{cheb} = -\cos\left(\frac{(j-1)\pi}{m-1}\right)$ is the j -th Chebyshev node. This mapping takes a set of m Chebyshev nodes as input and outputs a set of m transformed nodes. For $\alpha = 1$, the KTE map outputs equispaced nodes while as $\alpha \rightarrow 0$, the mapping approaches the identity mapping. By selecting different values of $\alpha \in (0, 1)$, we adjust the clustering of the nodes near the boundaries of the interval. As a result, this mapping gives us another way to reduce the dimension of the optimization space even further.

Adjusting the clustering of the boundary nodes can now be achieved with the selection of just one parameter whereas in the centered finite difference method, the dimension of the optimization space was equal to the size of the finite difference stencil. [35, 36, 45] have also leveraged the KTE mapping for finite difference schemes. In these works, the KTE mapping was used to scale the eigenvalue spectrum of spatial discretization matrices.

Again, we optimize using the Lebesgue constant as the objective function. Optimizing the single parameter, $\alpha \in (0, 1)$, is much faster than the partial centered finite difference method. We revisit the previous two examples: a 9 point stencil with 50 sampling points and a 15 point stencil with 1000 sampling points. The results are shown in Figure 4.7. In these examples, we obtain Lebesgue constants of $\Lambda = 1.8815$ and $\Lambda = 2.4719$. Again, we notice that the minimizing the Lebesgue constant leads to stable differentiation matrices. We also observe the overall structure of the nodes. Since there is only parameter, the structure of the sampling nodes is limited. The KTE mapping will not be able to generate points with the spacing shown in Figures 4.5 and 4.6 due to the limited degrees of freedom.

In Figure 4.8, we plot the optimal α values for varying stencil sizes and varying number of sampling points. We notice that as we increase the order of the finite difference method, more stretching near the boundary is required. The results in Figure 4.8, very closely resemble the results from [35, 36, 45]. These works looked for the optimal α values needed to stretch the sampling nodes in order to move the eigenvalue spectrum into the relevant stability regions. The similarity of results in this work and in [35, 36, 45] suggest the relation between the Lebesgue constant and stable spatial differentiation matrices.

Although the KTE map allows us to find optimal sampling points for finite difference methods in a fast and efficient manner, it can only be implemented in 1D.

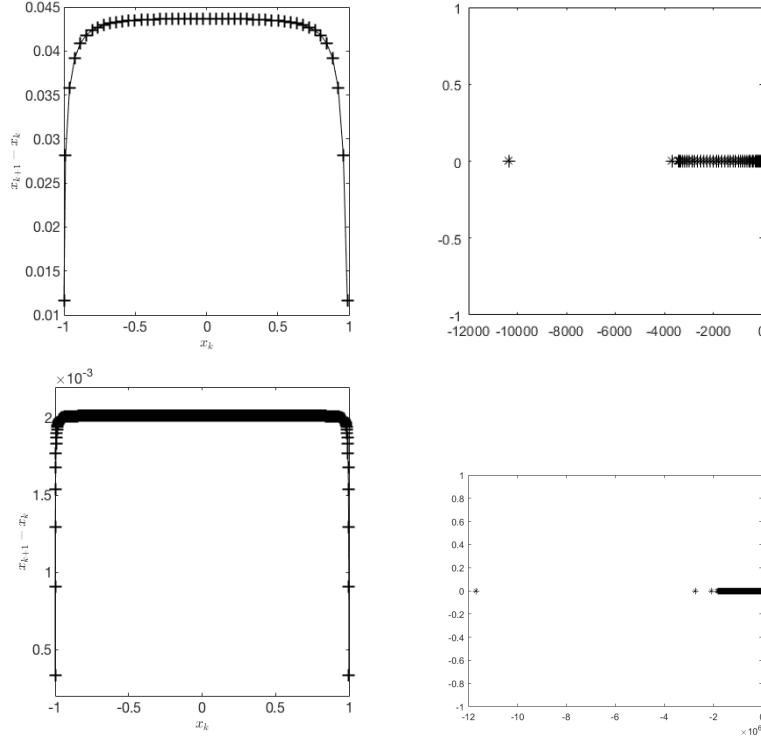


Figure 4.7: KTE Example. *Top Left:* Spacing of optimal sampling points for 50 sampling points with a 9 point stencil. *Top Right:* The eigenvalues of the resulting differentiation matrix in the real/imaginary plane. *Bottom Left:* Spacing of optimal sampling points for 1000 sampling points with a 15 point stencil. *Bottom Right:* The eigenvalues of the resulting differentiation matrix in the real/imaginary plane.

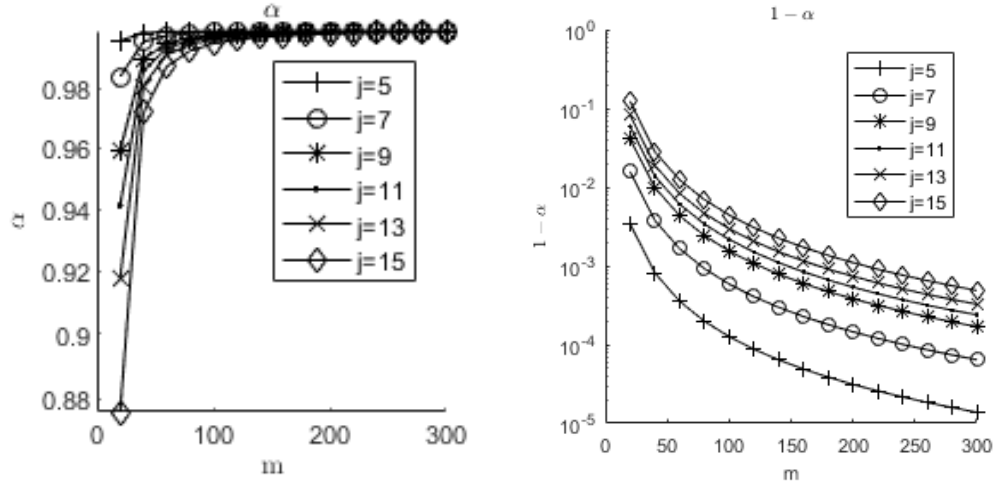


Figure 4.8: Optimal α Values for Various Nodal Set and Stencil Sizes. *Left:* Linear plot of α . *Right:* Semilog plot of $1 - \alpha$.

The KTE map will not be able to be applied in 2D due to the nature of the complex regions. This motivates the need for a robust method for finding optimal sampling points for complex 2D regions.

4.1.5 *Column-Pivoting Method*

We recall that the Column-pivoting QR Algorithm provides a robust method for selecting near-optimal points, called approximate-Fekete points. For finding sampling points for finite difference methods, this algorithm is free of the structural dependencies shown in the two dimension reduction methods discussed so far. To implement this algorithm, the only requirements are a set of candidate points and a choice of basis. The first requirement can be satisfied with the density function approach discussed in Section 3.1.3 and the second requirement can be satisfied by using the Chebyshev basis. We could also consider RBFs; however, this will be discussed in the next section.

We consider again the problem with $m = 50$ sampling points and a $j = 9$ point stencil. We plot the resulting spacing of sampling points obtained from the Column-pivoting QR Algorithm in Figure 4.9. In this case, the resulting Lebesgue constant is $\Lambda = 1.97057$ compared to $\Lambda = 1.8538$ for the Partial Centered Finite Difference method and $\Lambda = 1.8815$ for the KTE method. We notice the similarity in the spacing between the points from the Column-pivoting QR Algorithm and the Partial Centered Finite Difference method in Figure 4.5. We see that the Lebesgue constant is marginally worse; however, we stress the fact that this method does not impose any structural requirements upon the sampling nodes. As a result, this method may provide a robust algorithm for complex regions.

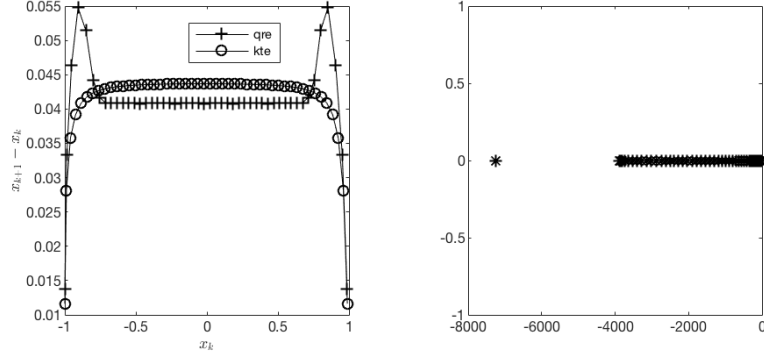


Figure 4.9: Finite Difference Column-pivoting QR example. *Left:* Spacing of optimal sampling points for the example with 50 sampling points and a 9 point stencil. *Right:* The eigenvalues of the resulting differentiation matrix in the real/imaginary plane.

4.2 RBF Finite Difference (RBF-FD) Methods in 1D

We also look to extend the finite difference methods to implement RBFs. Thus far, the optimal sampling points we found have been for piecewise polynomials. We will look to find optimal sampling points when we use RBFs as the basis rather than polynomials. The ultimate goal will be to find optimal sampling points for RBF-FD methods for complex 2D regions.

Using the system in Equation 3.14, we derive the differentiation matrix, $\mathbf{D}$.

$$Ls(x_c) = \sum_{k=0}^n c_k L\phi(\|x_c - x_k\|) \quad (4.11)$$

$$= \begin{bmatrix} L\phi(\|x_c - x_0\|) & L\phi(\|x_c - x_1\|) & \dots & L\phi(\|x_c - x_n\|) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \quad (4.12)$$

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \phi(\|x_0 - x_0\|) & \phi(\|x_0 - x_1\|) & \dots & \phi(\|x_0 - x_n\|) \\ \phi(\|x_1 - x_0\|) & \phi(\|x_1 - x_1\|) & \dots & \phi(\|x_1 - x_n\|) \\ \phi(\|x_2 - x_0\|) & \phi(\|x_2 - x_1\|) & \dots & \phi(\|x_2 - x_n\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|x_n - x_0\|) & \phi(\|x_n - x_1\|) & \dots & \phi(\|x_n - x_n\|) \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad (4.13)$$

$$= \mathbf{A}^{-1} \mathbf{y}, \quad (4.14)$$

$$Ls(x_c) = \begin{bmatrix} L\phi(\|x_c - x_0\|) & L\phi(\|x_c - x_1\|) & \dots & L\phi(\|x_c - x_n\|) \end{bmatrix} \mathbf{A}^{-1} \mathbf{y}, \quad (4.15)$$

$$= \mathbf{d}_c \mathbf{y} \quad (4.16)$$

$\mathbf{d}_c$ is one row of the differentiation matrix, $\mathbf{D}$. Repeating the process in Equations 4.12-4.16 for all points x_c in the region obtains the full differentiation matrix. RBF interpolation, and thus, RBF finite difference methods can also be formulated in a piecewise manner. This done by again by selecting a stencil size and generating the appropriate RBF Vandermonde matrix with regard to each stencil.

For polynomial interpolation, the Vandermonde matrix can be tabulated after creating a set of candidate points and choosing the basis. With RBFs, the interpolation matrix in Equation 3.14 can only be tabulated once we have the location of the centers, $(x_0, \dots, x_n)$. Thus, if we wanted to implement the Column-pivoting QR Algorithm again, we must first generate a set of candidate points, select a basis, and make a starting guess of sampling points in order to tabulate interpolation matrices.

For the RBF-FD in 1D, we choose to use the Lagrange polynomials as a basis. With a set of candidate points and a starting guess, we then implement the Column-pivoting QR Algorithm on the matrix L , defined in Equation 4.17 in order to minimize the Lebesgue constant. Since the RBF-FD methods are local approximations, we calculate the Lagrange polynomials in a piecewise manner as previously depicted in Figure 4.2. Furthermore, we can iteratively apply the Column-pivoting QR Algorithm

to ensure we minimize the Lebesgue constant in the case we make a poor starting guess.

$$L = \begin{bmatrix} l_1(x_0, y_0) & l_2(x_0, y_0) & \dots & l_n(x_0, y_0) \\ l_1(x_1, y_1) & l_2(x_1, y_1) & \dots & l_n(x_1, y_1) \\ l_1(x_2, y_2) & l_2(x_2, y_2) & \dots & l_n(x_2, y_2) \\ \vdots & \vdots & \ddots & \vdots \\ l_1(x_m, y_m) & l_2(x_m, y_m) & \dots & l_n(x_m, y_m) \end{bmatrix} \quad (4.17)$$

We consider again the problem with $m = 50$ sampling points and a $j = 9$ point stencil. We plot the resulting spacing of sampling points obtained from the Column-pivoting QR Algorithm using the Gaussian RBF basis in Figure 4.10. The resulting Lebesgue constant is $\Lambda = 1.7480$. One important aspect when using the RBF basis is selecting the proper shape parameter, ξ . When using the RBF basis, the shape parameter affects both the accuracy and the condition number of Vandermonde matrices. It is well known that the accuracy and condition number cannot be both kept small [33, 34], and so, it is important to consistently select the shape parameter to balance accuracy and conditioning [14, 27]. In the examples discussed in this text, the shape parameter is chosen such that the condition number of the Vandermonde matrices remain near $\kappa = 1 \times 10^{10}$.

4.3 RBF-FD Methods in 2D

4.3.1 RBF-FD Methods with Polyharmonic Splines and Polynomials Background

Introduction

In [2, 3], Polyharmonic Splines (PHS) and polynomials were combined to calculate RBF-FD differentiation matrices. One of the key benefits of combining PHSs with

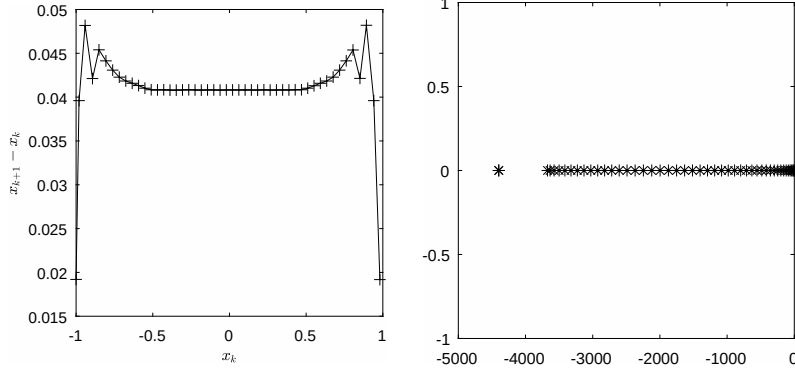


Figure 4.10: Finite Difference Column-pivoting QR Example with RBFs. *Left:* Spacing of optimal sampling points for the example with 50 sampling points and a 9 point stencil. *Right:* The eigenvalues of the resulting differentiation matrix in the real/imaginary plane.

polynomials was the fact that high-order accuracy could be obtained from resulting RBF-FD interpolation matrices. These matrices benefited from the improved conditioning that resulted from appending the polynomials. Another major benefit of these methods was the elimination of the requirement to select optimal shape parameters. We observe that in Table 3.3, all of the RBFs, with the exception of PHSs, requires the selection of a shape parameter, ξ . As a result, the need to balance accuracy and conditioning through the tuning of the shape parameter becomes a problem itself. The use of PHSs with polynomials eliminates this requirement. Instead of having to select shape parameters to handle different resolutions, the degree of the PHS is preselected and remains constant.

In [2, 3], RBF-FD methods are considered for simple 2D regions such as the unit and square. The sampling node layouts for these regions are uniform. For example, they can be equispaced in 2D, taking on a Cartesian form, or they can be distributed hexagonally as shown in Figure 4.11. We will show that using the same Column-pivoting QR Algorithm, we can find good nodes for RBF-FD methods combining PHSs and polynomials for complex regions in 2D. The method we propose provides a robust algorithm for finding good nodes for general complex regions. Furthermore,

they display the same behavior in terms of accuracy when compared to the results shown in [2, 3], and build upon those results by providing differentiation matrices with improved stability. The node finding algorithm also mitigates some of the crucial constraints discussed in [2, 3] which we will mention below.

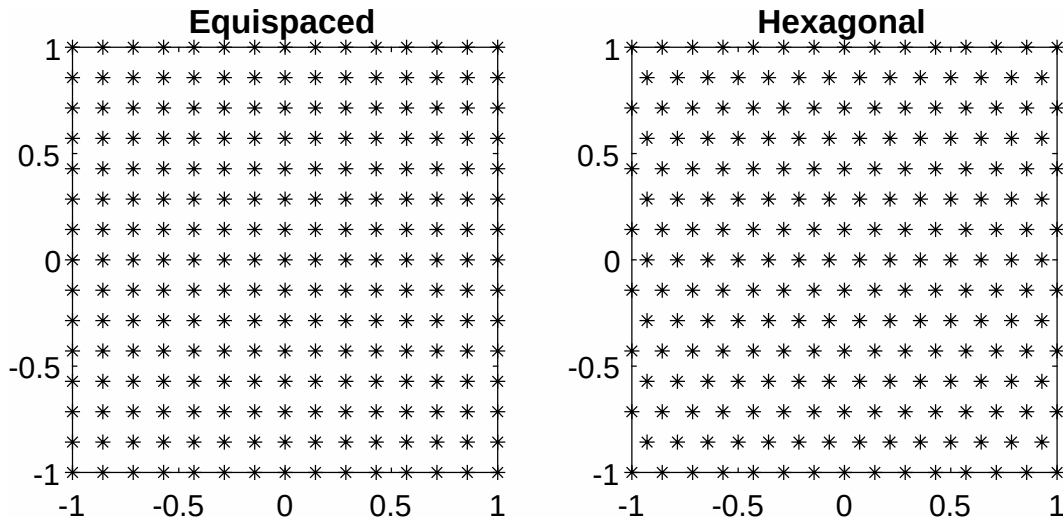


Figure 4.11: Square Shape Equispaced and Hexagonal Node Layout

In the 2D finite difference case, we can find which nodes are used in the finite-difference weight calculations using MATLAB's *KDTree* and *knnsearch* functions. To calculate the RBF-FD weights at a given point, a stencil size is chosen and the nearest neighbors are found. Figure 4.12 below illustrates an example of what these stencils should look like in a complex 2D region such as the Bumped-disk shape. The sampling points are marked by the dots while the point at which we calculate the RBF-FD weights is marked by an asterisk with the relevant stencil points being outlined circles.

Calculating RBF-FD Weights

Suppose we wish to calculate the differentiation weights, $w_0, \dots, w_n$, for the point $x = x_c$ using an $n + 1$ point stencil. The new linear system is depicted in Equation

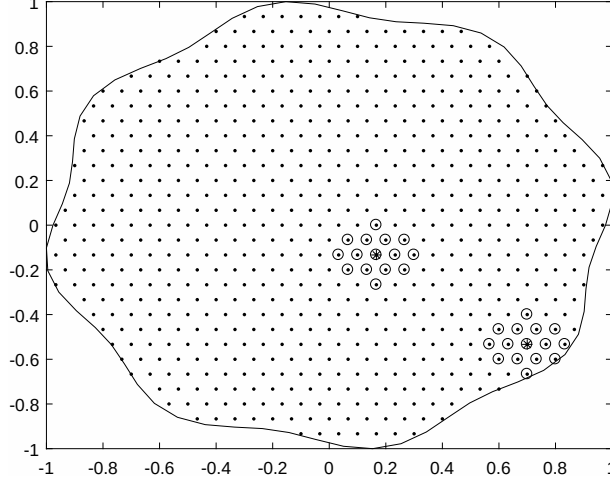


Figure 4.12: 15 Node Stencil Example for the Bumped-disk Region

4.18 below.

$$\begin{bmatrix} \|x_0 - x_0\|_2^{2m-1} & \dots & \|x_0 - x_n\|_2^{2m-1} & 1 & x_0 & y_0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \|x_n - x_0\|_2^{2m-1} & \dots & \|x_n - x_n\|_2^{2m-1} & 1 & x_n & y_n \\ \hline 1 & \dots & 1 & 0 & 0 & 0 \\ x_0 & \dots & x_n & 0 & 0 & 0 \\ y_0 & \dots & y_n & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_0 \\ \vdots \\ w_n \\ w_{n+1} \\ w_{n+2} \\ w_{n+3} \end{bmatrix} = \begin{bmatrix} L\|x - x_0\|_2^{2m-1}|_{x=x_c} \\ \vdots \\ L\|x - x_n\|_2^{2m-1}|_{x=x_c} \\ L1|_{x=x_c} \\ Lx|_{x=x_c} \\ Ly|_{x=x_c} \end{bmatrix} \quad (4.18)$$

The top-left sub-matrix is the usual RBF interpolation matrix. We see that a Vandermonde matrix consisting of the same stencil points, but using a monomial basis, is appended to the RBF interpolation matrix. In this case, the PHSs are combined with polynomials up to first degree.

One important constraint mentioned in [2, 3] is the requirement for approximately twice the number of stencil nodes as there are polynomial basis functions. The example given in [3] states that for a 37 node stencil, only polynomials of degree up to 4 (15 basis functions) should be used. This requirement ensures the well-conditioning

of the matrix in the left-hand side of equation 4.18.

One drawback of using strictly RBFs is the fact that, as we use more nodes for the finite-difference schemes, the resolution increases causing the user to have to increase the shape parameter in order to maintain the well-conditioning of the interpolation matrix. As a result, the need to keep the matrix well-conditioned cancels out any increase in accuracy resulting from using more sampling nodes in the RBF-FD scheme. Furthermore, the use of a polynomial basis does not work well for finite difference methods in 2D. In this case, the interpolation matrices quickly become ill-conditioned. Using RBF-FD methods with PHSs in conjunction with polynomials solves both of the issues that arise from using RBFs and polynomials individually. With PHSs and polynomials, the need to select optimal shape parameters is eliminated along with the need to sacrifice accuracy for conditioning. Furthermore, by appending additional basis vectors to the interpolation matrix by adding polynomials (see Equation 4.18), we keep the interpolation matrix well-conditioned as long as we follow the basic constraint of requiring twice the number of nodes in the stencil as there are polynomial basis functions.

Accuracy Considerations

It was shown in [2, 3] that the convergence rate of RBF-FD methods combining PHSs and polynomials depends on the degree of polynomials used. The rate of convergence does not depend on the parameter, m , which defines the PHS. For example, approximations of first order derivatives converge at the rate of $O(h^p)$ where h is the spacing and p is the degree of polynomials used in the RBF-FD method. For Laplacian approximations, we can expect the rate of convergence to be $O(h^{p-1})$. Figure 4.13 below depicts an example of the convergence rate these RBF-FD methods provide. In this case, a hexagonal nodal set is used on the unit square as shown in Figure 4.11.

A 51 point stencil is used such that there are enough nodes in the stencil to handle the inclusion of polynomials up to degree $p = 5$. The PHS used is $\phi(r) = r^3$. We plot the relative error of the approximation of $\frac{d}{dx}(1 + \sin(4x) + \cos(3x) + \sin(2y))$ against the spacing, h . We also plot the expected convergence rate for each degree of polynomials used in dashed lines.

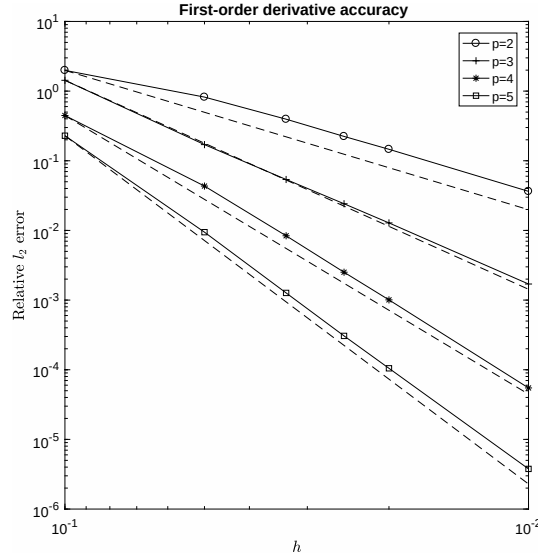


Figure 4.13: Convergence Rates of a First-Order Derivative Approximation using PHS and Polynomials

Eigenvalue Stability Considerations

For RBF-FD methods using PHSs and polynomials, we again require differentiation matrices with suitable eigenvalues for time-stepping. In Figure 4.14 below, we plot the Dirichlet eigenvalues obtained from calculating the Laplacian approximant using $\phi(r) = r^3$ with fourth degree polynomials and again, a 51 point stencil. The spacing is $h = .1$. We notice that the eigenvalues do not lie on the negative real axis. As a result, hyperviscosity must be applied in order to provide stable time-stepping if we wish to consider PDEs such as the wave equation.

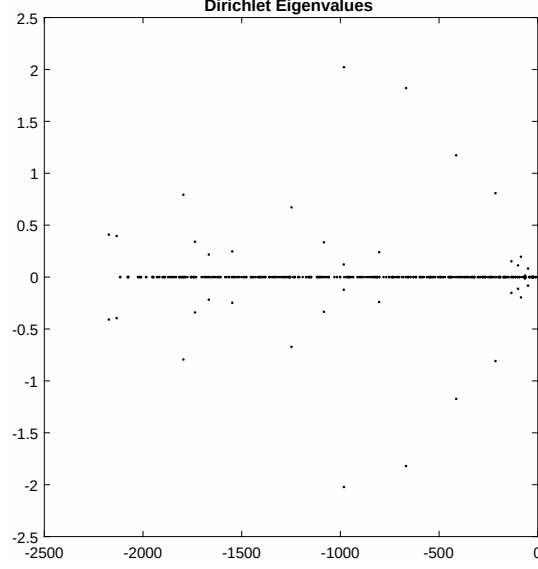


Figure 4.14: Dirichlet Eigenvalues for Approximation using PHS and Polynomials

4.3.2 Applying the Column-Pivoting QR Method

In order to find optimal sampling points for RBF-FD methods using PHS and polynomials, we again use the Column-pivoting QR Algorithm. The idea is similar to the 1D RBF-FD case where we apply the algorithm iteratively to the piecewise-constructed Lagrange polynomial matrix. We build the Lagrange polynomial matrix by solving for the Lagrange polynomial coefficients (polynomial degree, $p = 1$, in this case) in the system in Equation 4.19 below.

$$\left[\begin{array}{ccc|ccc}
 \|x_0 - x_0\|_2^{2m-1} & \dots & \|x_0 - x_n\|_2^{2m-1} & 1 & x_0 & y_0 \\
 \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
 \|x_n - x_0\|_2^{2m-1} & \dots & \|x_n - x_n\|_2^{2m-1} & 1 & x_n & y_n \\
 \hline
 1 & \dots & 1 & 0 & 0 & 0 \\
 x_0 & \dots & x_n & 0 & 0 & 0 \\
 y_0 & \dots & y_n & 0 & 0 & 0
 \end{array} \right] \left[\begin{array}{ccc}
 c_{0,0} & \dots & c_{0,n} \\
 \vdots & \vdots & \vdots \\
 c_{n,0} & \dots & c_{n,n} \\
 \hline
 c_{n+1,0} & \dots & c_{n+1,n} \\
 c_{n+2,0} & \dots & c_{n+2,n} \\
 c_{n+3,0} & \dots & c_{n+3,n}
 \end{array} \right] = \left[\begin{array}{c}
 \mathcal{I} \\
 \mathbf{0}
 \end{array} \right] \quad (4.19)$$

Once we calculate the Lagrange coefficients for each stencil, we can populate the Lagrange polynomial matrix for each candidate point, $(x_0, y_0), \dots, (x_m, y_m)$, shown in Equation 4.20 below. The Column-pivoting QR Algorithm is then applied to this matrix, L , iteratively in order to find optimal sampling points. The goal is again to select the points resulting from the Column-pivoting QR Algorithm with the lowest Lebesgue constant.

$$L = \begin{bmatrix} l_1(x_0, y_0) & l_2(x_0, y_0) & \dots & l_n(x_0, y_0) \\ l_1(x_1, y_1) & l_2(x_1, y_1) & \dots & l_n(x_1, y_1) \\ l_1(x_2, y_2) & l_2(x_2, y_2) & \dots & l_n(x_2, y_2) \\ \vdots & \vdots & \ddots & \vdots \\ l_1(x_m, y_m) & l_2(x_m, y_m) & \dots & l_n(x_m, y_m) \end{bmatrix} \quad (4.20)$$

4.3.3 Results in 2D

To apply the Column-pivoting QR Algorithm, we recall that a starting guess is required to first populate the Lagrange matrix. Since we will iteratively apply the QR algorithm in order to improve the Lebesgue constant, we can make starting guesses for complex shapes using node layouts similar to those used for the unit square depicted in Figure 4.11.

Optimal Sampling Points and Eigenvalue Stability

For the Bumped-disk shape, we first populate the unit square hexagonal nodes, draw the Bumped-disk shape, and take the hexagonal square nodes which lie within the boundary of the Bumped-disk. Along with these points that fall within the boundary, we also choose to select a number of boundary points to include in the starting guess. Then, from this starting guess, we apply the Column-pivoting QR Algorithm on a

set of candidate points which discretize the Bumped-disk, including its boundary so the QR algorithm can provide optimal sampling points for both the interior and the boundary of the complex region. In Figure 4.15, we plot both the starting guess and the resulting optimal sampling nodes for the Bumped-disk region. This case considers the Bumped-disk region using a 37 point stencil, $\phi(r) = r^3$, and polynomials up to degree $p = 4$. In this example, 734 nodes are used for RBF-FD calculations.

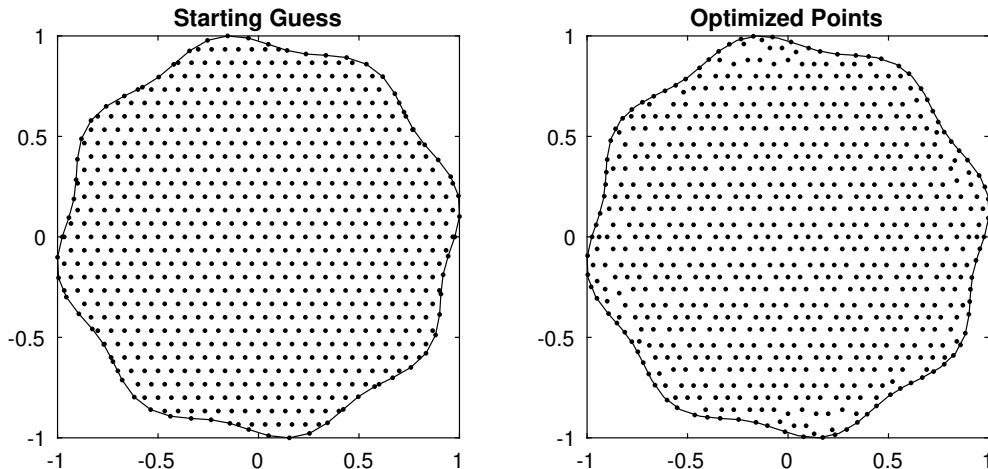


Figure 4.15: Starting Guess Points and Optimal Samplings Points for the Bumped-disk Region

In this case, the Column-pivoting QR Algorithm decreased the Lebesgue constant from $\Lambda = 8.99$ to $\Lambda = 4.40$. When considering the Dirichlet eigenvalues, we notice that the resulting optimal sampling points provides much better eigenvalues for time-stepping, depicted in Figure 4.16 below. When considering stability regions in the λt plane, the eigenvalues of the differentiation matrix from the optimal sampling points allow for a 342% increase in time-step. Furthermore, the eigenvalues are also much improved in the magnitude of the imaginary part. As a result, it may be easier to apply hyperviscosity methods in order to ensure stable time-stepping.

Figure 4.17 displays the results for the Peanut region. This example considers a 37 point stencil, $\phi(r) = r^3$, and polynomials up to degree $p = 4$. Here, 383 nodes are used for the RBF-FD calculations. We see similar improvements in the differentiation

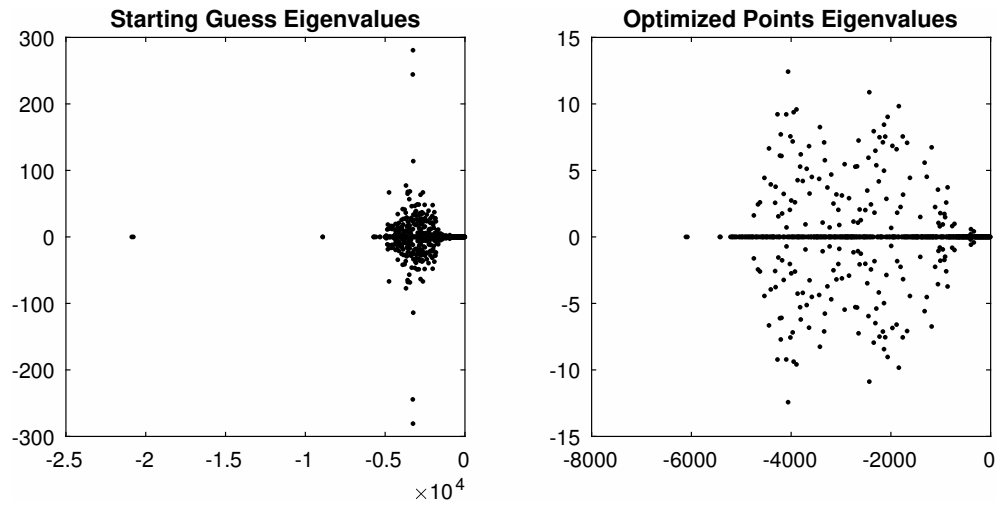


Figure 4.16: Differentiation Matrix Eigenvalues for Starting Guess Points and Optimized Points

matrix eigenvalues.

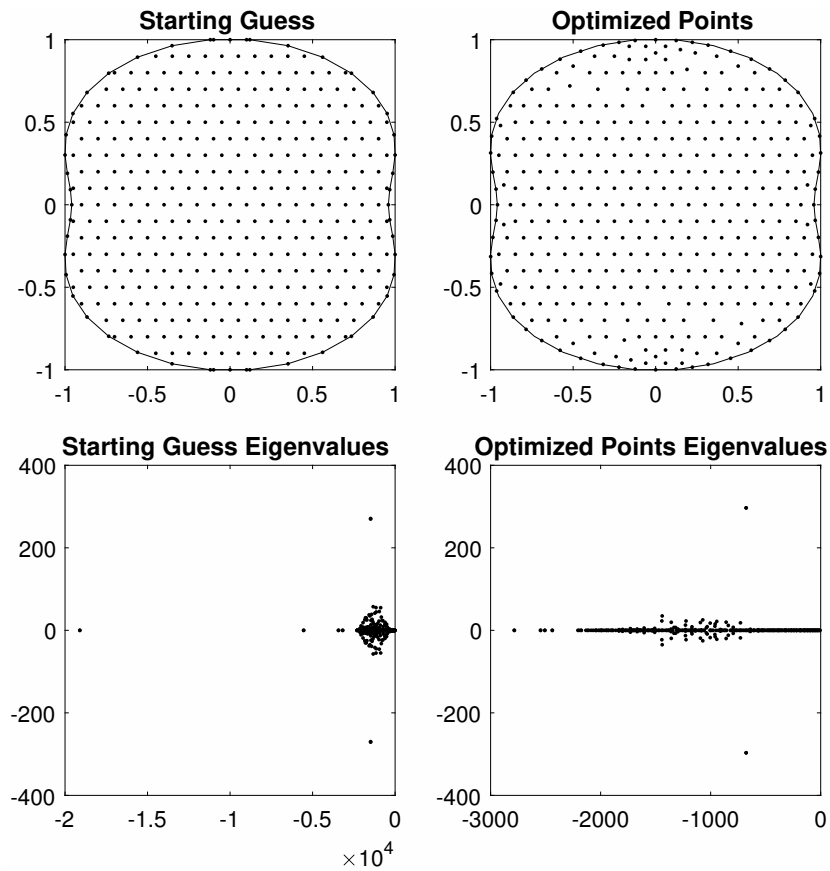


Figure 4.17: Starting Guess Points and Optimal Samplings Points for the Peanut

In some cases, the resulting optimized sampling nodes can even eliminate the need to apply hyperviscosity methods. Consider the Bumped-disk region, using a 37 point stencil with $\phi(r) = r^3$ and polynomials up to degree $p = 2$ with 96 sampling nodes. The eigenvalues for the optimized points, shown in Figure 4.18 lie on the negative real axis whereas the eigenvalues for the starting guess are complex.

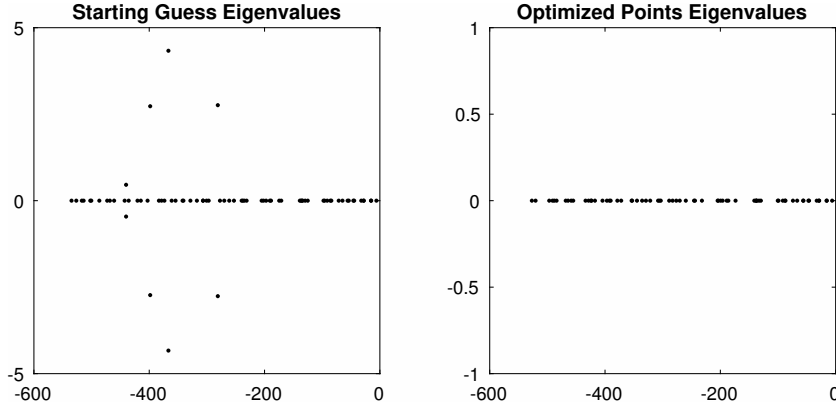


Figure 4.18: Eigenvalue Improvement for Bumped-disk Region

Accuracy Considerations

Along with eigenvalue stability, we would also like to keep the accuracy of differential operator approximation in consideration. In Figure 4.19, we plot the error obtained from approximating $\frac{d}{dx}(1 + \sin(4x) + \cos(3x) + \sin(2y))$ with both starting guesses and the optimized sampling points for the Bumped-disk and Peanut region. We notice that with the optimized points, the accuracy is either better or the same in all cases.

Node Stencil Requirement Improvement

One of the key constraints noted for using RBF-FD methods with PHSs and polynomials was the requirement for there to be at least a specific number of stencil nodes as compared to the degree of the polynomials used. With optimized sampling points, we can mitigate the consequences of this constraint. For example, consider

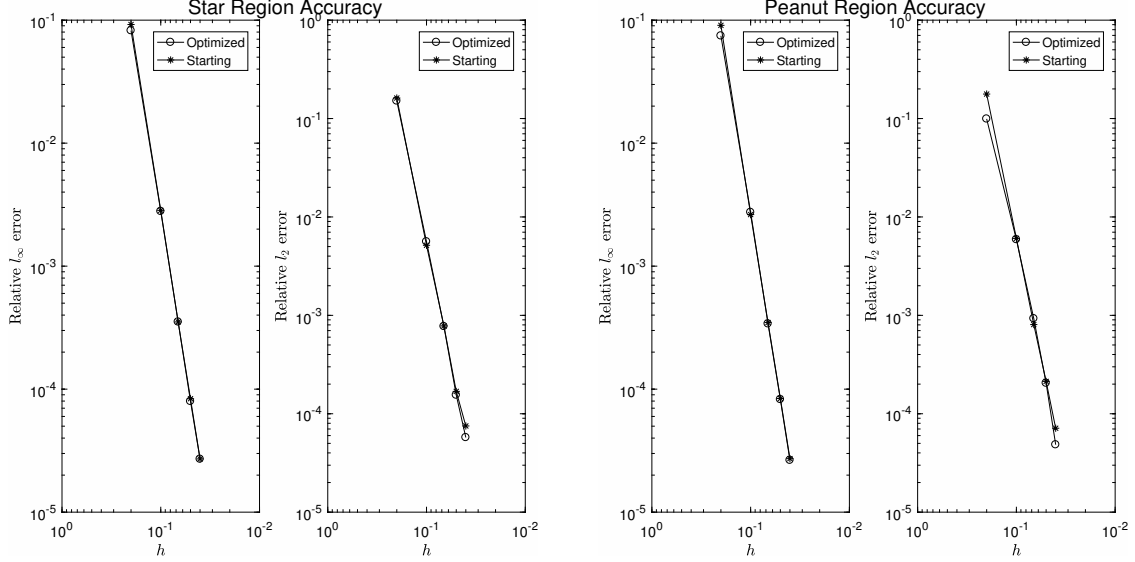


Figure 4.19: Sampling Node Accuracy Comparison for the Bumped-disk and Peanut Region

the Peanut region, using a 21 point stencil with $\phi(r) = r^3$ and polynomials up to degree $p = 4$ with 383 sampling nodes. In this case, it is recommended that there are at least 30 nodes in the stencil in order to maintain the well-conditioning of the interpolation matrices. The consequences of this constraint can be seen in Figure 4.20. Using the hexagonally distributed starting guess, we see that the Dirichlet eigenvalues cross over into the positive real plane, rendering the differentiation matrix unstable for time-stepping. Contrastingly, after we find the optimized sampling point using the Column-pivoting QR Algorithm, the eigenvalues move back to the negative real plane.

Not only do the optimized sampling points improve eigenvalue stability for stencils that do not contain the requisite number of points, they can also help improve in the accuracy of differential operator approximation. Consider the Peanut region, using a 27 point stencil with $\phi(r) = r^3$ and 383 sampling nodes. Using polynomials up to degree $p = 5$, the initial starting points are not well suited to approximate $\frac{d}{dx}(1 + \sin(4x) + \cos(3x) + \sin(2y))$. In this case, we obtain a relative error of

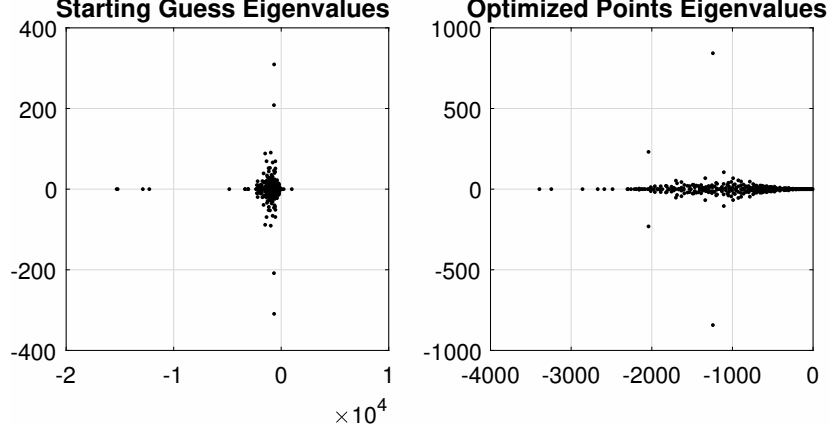


Figure 4.20: Eigenvalue Improvement for Peanut Region

$l_\infty = 34.18$ and $l_2 = 3.19$. After optimizing the points using the Column-pivoting QR Algorithm, these errors are reduced to $l_\infty = 2.70 \times 10^{-3}$ and $l_2 = 7.86 \times 10^{-4}$. Thus, we see that the optimized points can mitigate the stencil node requirements for both eigenvalue stability and accuracy.

4.3.4 Solving PDEs Using RBF-FD Methods with Polyharmonic Splines and Polynomials

We look to solve the same PDEs we used for constrained approximation in Section 3.2.5 using these RBF-FD Methods with PHSs and polynomials. After using the Column-pivoting QR algorithm to find optimal samplings points and differentiation matrices for the Peanut and the Bumped-disk regions, we find the solution, $u(t, x, y)$, at time $t = 10$ for the PDE:

$$\frac{\partial u}{\partial t} = \Delta u + \sin(t)$$

$$u_0 = \mathbf{0}.$$

To do this, we again implement node refinement. Figure below plots the accuracy results we obtain.

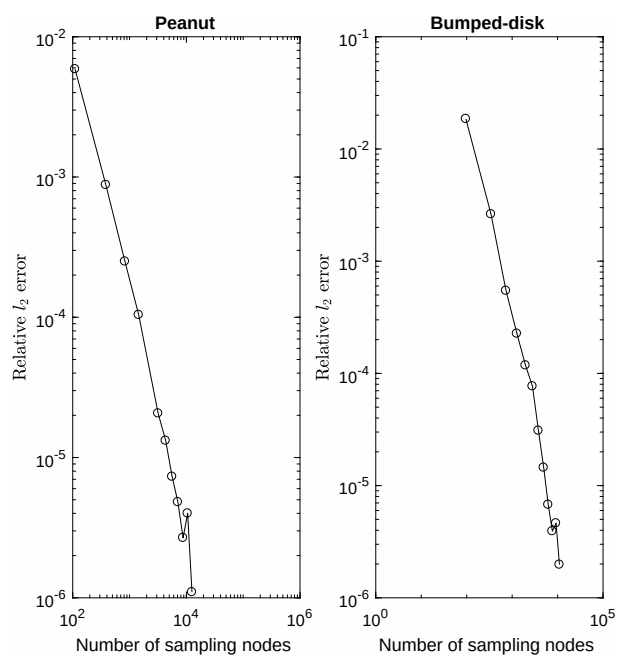


Figure 4.21: PDE Accuracy For RBF-FD Methods in 2D

Chapter 5

FINAL REMARKS

5.1 Ongoing Work

There are many ways to extend the work accomplished in the dissertation. One key area of focus for the future is making the Column-pivoting QR Algorithm more efficient for large problems. We recall that if we want to find a large number of optimal sampling points, we need require a very fine discretization in order to be able to correctly cluster the sampling points. As a result, the Vandermonde matrix in which we apply the Column-pivoting Algorithm grows very large, and with $\mathcal{O}(n^3)$ operations required, this problem quickly becomes computationally intractable. One way to handle this large problem is to split it into a set of smaller problems. This can be accomplished by applying partition of unity methods. This would allow us to partition the complex regions in which we would have to repetitively solve much smaller problems using the Column-pivoting QR Algorithm.

Finally, we note that in [17, 41], fast node generation algorithms have been proposed for 2D and 3D meshfree approximations. While these algorithms scale well with the number of nodes, they require the node density function to be known in advance. A possible line of research is to use the algorithms presented in this dissertation to estimate the optimal node density function by computing a moderate set of optimal nodes and then use the node generation algorithms in [17, 41] to calculate larger sets of points.

Another area to continue future research lies using constrained approximation to solve PDEs. In Chapter 3, we applied the Column-pivoting QR Algorithm to obtain

sampling points for constrained approximation to solve a few simple PDEs. We plan on applying these sampling points and their differentiation matrices to solve problems such as the Navier-Stokes equations. We also plan to consider applying hyperviscosity methods where needed.

5.2 Conclusions

This work focused on finding optimal sampling points for various approximation problems. In Chapter 3, we were able to develop a Greedy Algorithm in order to provide a way to minimize the Lebesgue constant for polynomial data fitting methods. The algorithm's ability to generate good points has been demonstrated, finding optimal sampling points for 2D regions such as the Peanut, Bumped-disk, Bumped-disk Ring and L-shape. This method was robust in that it found points with Lebesgue constant close to the minimal Lebesgue constant while still being computational feasible. We demonstrated the ability to take non-optimal starting sets and improve the optimality by adding points two at a time. This Greedy Algorithm also included a few speed-up modifications to even further lessen the computational requirements. In Chapter 3, we were also able to modify a well-known Column-pivoting QR Algorithm, used to find approximate-Fekete points for polynomial interpolation, to handle constrained approximation used to solve PDEs. Lastly, we used the Column-pivoting QR Algorithm to find optimal points for RBF data fitting methods. We demonstrated that the Column-pivoting QR Algorithm could be modified to be able to simultaneously select centers as well as sampling points for RBF interpolation.

In Chapter 4, we explored optimal sampling points for high-order finite difference methods. This was done for 1D as well as for 2D complex regions. In 1D, we first considered using piecewise polynomials to setup the problem of minimizing the Lebesgue constant. Once this was done, we applied three methods to find optimal

sampling points. The first two, the Partial Centered Finite Difference method and the KTE method both reduced the dimension of the optimization space. In the last method, we again used the Column-pivoting QR Algorithm to find optimal points. We showed that by minimizing the Lebesgue constant, we were able to find optimal sampling points with similar clustering that led to stable differentiation matrices for time-stepping. We also showed in this chapter how to consider optimal sampling points for RBF-FD methods. In particular, we demonstrated that we could find optimal points for RBF-FD methods which combined PHSs and polynomials. These methods had previously considered simple shapes with simple node layouts (Cartesian or Hexagonal). By using the Column-pivoting QR Algorithm, we could find optimal points for these RBF-FD methods on complex shapes while preserving accuracy and improving eigenvalue stability. These points also improved on established stencil node requirements for these RBF-FD methods.

REFERENCES

- [1] How globalsearch and multistart work - matlab and simulink. <https://www.mathworks.com/help/gads/how-globalsearch-and-multistart-work.html>.
- [2] Gregory Barnett. A robust RBF-FD formulation based on polyharmonic splines and polynomials. 2015.
- [3] Gregory A Barnett, Natasha Flyer, and Louis J Wicker. An RBF-FD polynomial method based on polyharmonic splines for the navier-stokes equations: Comparisons on different node layouts. *arXiv preprint arXiv:1509.02615*, 2015.
- [4] Victor Bayona, Natasha Flyer, Bengt Fornberg, and Gregory A Barnett. On the role of polynomials in RBF-FD approximations: Ii. numerical solution of elliptic pdes. *Journal of Computational Physics*, 332:257–273, 2017.
- [5] Len Bos, Jean-Paul Calvi, Norman Levenberg, Alvis Sommariva, and Marco Vianello. Geometric weakly admissible meshes, discrete least-squares approximations and approximate Fekete points. *Mathematics of Computation*, 80(275):1623–1638, 2011.
- [6] Len Bos, Stefano De Marchi, Alvis Sommariva, and Marco Vianello. Computing multivariate Fekete and Leja points by numerical linear algebra. *SIAM Journal on Numerical Analysis*, 48(5):1984–1999, 2010.
- [7] Len Bos and Norman Levenberg. On the calculation of approximate Fekete points: the univariate case. *Electronic Transactions on Numerical Analysis*, 30:377–397, 2008.
- [8] Matteo Briani, Alvis Sommariva, and Marco Vianello. Computing Fekete and Lebesgue points: Simplex, square, disk. *Journal of Computational and Applied Mathematics*, 236(9):2477–2486, 2012.
- [9] Marco Caliari, Stefano De Marchi, and Marco Vianello. Bivariate polynomial interpolation on the square at new nodal sets. *Applied Mathematics and Computation*, 165(2):261–274, 2005.
- [10] Marco Caliari, Stefano De Marchi, and Marco Vianello. Bivariate lagrange interpolation at the padua points: Computational aspects. *Journal of Computational and Applied Mathematics*, 221(2):284–292, 2008.
- [11] Stefano De Marchi, Robert Schaback, and Holger Wendland. Near-optimal data-independent point locations for radial basis function interpolation. *Advances in Computational Mathematics*, 23(3):317–330, 2005.
- [12] Tobin A Driscoll and Lloyd N Trefethen. *Schwarz-Christoffel mapping*, volume 8. Cambridge University Press, 2002.

- [13] Gregory E Fasshauer. *Meshfree Approximation Methods with Matlab:(With CD-ROM)*, volume 6. World Scientific Publishing Co Inc, 2007.
- [14] Gregory E Fasshauer and Jack G Zhang. On choosing optimal shape parameters for RBF approximation. *Numerical Algorithms*, 45(1-4):345–368, 2007.
- [15] Natasha Flyer, Bengt Fornberg, Victor Bayona, and Gregory A Barnett. On the role of polynomials in RBF-FD approximations: I. interpolation and accuracy. *Journal of Computational Physics*, 321:21–38, 2016.
- [16] Natasha Flyer, Grady B Wright, and Bengt Fornberg. Radial basis function-generated finite differences: A mesh-free method for computational geosciences. *Handbook of geomathematics: Springer Reference*, 130, 2014.
- [17] Bengt Fornberg and Natasha Flyer. Fast generation of 2D node distributions for mesh-free PDE discretizations. *Computers & Mathematics with Applications*, 69(7):531–544, 2015.
- [18] Bengt Fornberg and Natasha Flyer. *A primer on radial basis functions with applications to the geosciences*. SIAM, 2015.
- [19] Max Gunzburger and Aretha L Teckentrup. Optimal point sets for total degree polynomial interpolation in moderate dimensions. *arXiv preprint arXiv:1407.3291*, 2014.
- [20] Ling Guo, Akil Narayan, Liang Yan, and Tao Zhou. Weighted approximate fekete points: Sampling for least-squares polynomial approximation. *arXiv preprint arXiv:1708.01296*, 2017.
- [21] Thomas Hagstrom and George Hagstrom. Grid stabilization of high-order one-sided differencing i: First-order hyperbolic systems. *Journal of Computational Physics*, 223(1):316–340, 2007.
- [22] Thomas Hagstrom and George Hagstrom. Grid stabilization of high-order one-sided differencing ii: second-order wave equations. *Journal of Computational Physics*, 231(23):7907–7931, 2012.
- [23] Miguel Hermanns and Juan Antonio Hernández. Stable high-order finite-difference methods based on non-uniform grid point distributions. *International journal for numerical methods in fluids*, 56(3):233–255, 2008.
- [24] Jan S Hesthaven. From electrostatics to almost optimal nodal sets for polynomial interpolation in a simplex. *SIAM Journal on Numerical Analysis*, 35(2):655–676, 1998.
- [25] Dan Kosloff and Hillel Tal-Ezer. A modified Chebyshev pseudospectral method with an $\mathcal{O}(n - 1)$ time step restriction. *Journal of Computational Physics*, 104(2):457–469, 1993.
- [26] Hassan Majidian. Automatic computing of oscillatory integrals. *Numerical Algorithms*, 77(3):867–884, 2018.

- [27] Michael Mongillo. Choosing basis functions and shape parameters for radial basis function methods. *SIAM Undergraduate Research Online*, 4(190-209):2–6, 2011.
- [28] Akil Narayan and Dongbin Xiu. Stochastic collocation methods on unstructured grids in high dimensions via interpolation. *SIAM Journal on Scientific Computing*, 34(3):A1729–A1752, 2012.
- [29] Akil Narayan and Dongbin Xiu. Constructing nested nodal sets for multivariate polynomial interpolation. *SIAM Journal on Scientific Computing*, 35(5):A.2293–A.2315, 2013.
- [30] Rodrigo B Platte. *Accuracy and stability of global radial basis function methods for the numerical solution of partial differential equations*. University of Delaware, 2005.
- [31] Gabriele Santin and Bernard Haasdonk. Convergence rate of the data-independent P-greedy algorithm in kernel-based approximation. *Dolomites Research Notes on Approximation*, 10, 2017.
- [32] Thomas Sauer and Yuan Xu. On multivariate lagrange interpolation. *Mathematics of Computation*, 64(211):1147–1170, 1995.
- [33] Robert Schaback. Error estimates and condition numbers for radial basis function interpolation. *Advances in Computational Mathematics*, 3(3):251–264, 1995.
- [34] Robert Schaback. A practical guide to radial basis functions. *Electronic Resource*, 11:1–12, 2007.
- [35] Ratnesh K Shukla, Mahidhar Tatineni, and Xiaolin Zhong. Very high-order compact finite difference schemes on non-uniform grids for incompressible Navier–Stokes equations. *Journal of Computational Physics*, 224(2):1064–1094, 2007.
- [36] Ratnesh K Shukla and Xiaolin Zhong. Derivation of high-order compact finite difference schemes for non-uniform grid using polynomial interpolation. *Journal of Computational Physics*, 204(2):404–429, 2005.
- [37] Simon J Smith. Lebesgue constants in polynomial interpolation. In *Annales Mathematicae et Informaticae*, volume 33, pages 1787–5021. Eszterházy Károly College, Institute of Mathematics and Computer Science, 2006.
- [38] Alvis Sommariva and Marco Vianello. Computing approximate Fekete points by QR factorizations of vandermonde matrices. *Computers & Mathematics with Applications*, 57(8):1324–1336, 2009.
- [39] Lloyd N Trefethen. *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, 2013.
- [40] Marc Van Barel, Matthias Humet, and Laurent Sorber. Approximating optimal point configurations for multivariate polynomial interpolation. *Electronic Transactions on Numerical Analysis*, 42:41–63, 2014.

- [41] Kiera van der Sande and Bengt Fornberg. Fast variable density 3D node generation. *arXiv preprint arXiv:1906.00636*, 2019.
- [42] Holger Wendland. *Scattered data approximation*, volume 17. Cambridge university press, 2004.
- [43] Daniel Wirtz and Bernard Haasdonk. A vectorial kernel orthogonal greedy algorithm. *Dolomites Research Notes on Approximation*, 6(Special_Issue), 2013.
- [44] Min Yang, Stefanie Biedermann, and Elina Tang. On optimal designs for nonlinear models: a general and efficient algorithm. *Journal of the American Statistical Association*, 108(504):1411–1420, 2013.
- [45] Xiaolin Zhong and Mahidhar Tatineni. High-order non-uniform grid schemes for numerical simulation of hypersonic boundary-layer stability and transition. *Journal of Computational Physics*, 190(2):419–458, 2003.