RICE UNIVERSITY

Adaptive Techniques Applied to the Sequentially Optimized Meshfree Approximation

by

Rachel Mittelman

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

Andrew J. Meade, Chair
Professor of Mechanical Engineering

J. Ed Akin
Professor of Mechanical Engineering

Ilinca Stanciulescu
Assistant Professor of Civil Engineering

Houston, Texas
April, 2014
ABSTRACT

Adaptive Techniques Applied to the Sequentially Optimized Meshfree Approximation

by

Rachel Mittelman

This thesis advances the meshless Sequentially Optimized Meshfree Approximation (SOMA) from a fixed grid to an adaptive one by applying residual-based adaptive techniques. In its fixed grid form, SOMA constructs an approximation of an equation solution using optimized radial basis functions (RBFs), but deletes the RBF parameters once each basis function is appropriately added. The first proposed method saves this information, constructs an approximation of the solution, and intelligently adds nodes to the problem domain. The second proposed method is a flexible interpolation scheme which does not require this basis saving technique, although the two techniques can be combined. When applied to various equations, these adaptive algorithms demonstrate the convergence required to achieve a satisfactory level of precision, saving time and computational effort for the same mathematical result as a denser grid. Applications of this algorithm include function approximation as well as differential equations which demonstrate its capability and robustness.
Acknowledgements

First, I must thank my amazing husband, Q, who has been a source of constant love and encouragement through some of the most academically challenging times of my life. Thank you to my wonderful family that has shaped who I am today. You inspire and challenge me- and for that I am grateful. I thank Dr. Meade for all of his tireless advice and academic support for the last two years. I also extend thanks to those in the lab who also have helped me far beyond what is expected. Thank you to Dr. Jan Hewitt for tirelessly sharing her linguistic expertise and helping me edit the following chapters. To my friends in Houston and around the world- you all have made this journey memorable. I wish all of you the best of luck in future endeavors and let’s all stay in touch as we soldier on.

Finally, I must sincerely thank Professor Akin and Professor Stanciulescu for taking time out of their busy schedules to review this thesis and my work.

Disclaimer

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.
Contents

Abstract ii
Acknowledgements ........................................ iii
List of Illustrations .................................. vii
Nomenclature and Abbreviations ......................... xi

1 Introduction ........................................ 1

2 Overview of Adaptive Techniques .................... 4
  2.1 Adaptive Finite Element Type Methods ............ 4
     2.1.1 Nodal Movement Techniques ................. 5
     2.1.2 Mesh Enrichment Techniques ................. 6
  2.2 Adaptive Collocation Methods .................... 9

3 The Sequentially Optimized Meshfree Approximation 12
  3.1 The Approximation ................................ 12
  3.2 Radial Basis Functions .......................... 13
  3.3 Residual Calculations ............................ 15
  3.4 Determining the Coefficients in SOMA .......... 17
  3.5 Enforcing Boundary Conditions .................. 17
  3.6 Calculating Derivatives in SOMA ............... 18
     3.6.1 Derivatives and Poisson’s Equation ........ 20
     3.6.2 Properties of Derivatives and Convergence in SOMA 23
  3.7 SOMA Applied to a Compressible Flow Problem ... 29
A Optimized Coefficients in SOMA 79
  A.1 Splitting the Coefficient ........................................ 79
  A.2 Results of Optimized Coefficients ............................ 80

B Gradient and Sparseness as Adaptation Metrics 84
  B.1 Gradient as an Adaptation Metric ............................ 84
  B.2 Sparseness as an Adaptation Metric .......................... 85

C Training and Test Nodes 91
  C.1 Function Approximation Example of Training and Test Nodes . . . . 91
  C.2 Problems with the Training and Test Node Method ............. 96
# Illustrations

2.1 Typical Flowchart for RBF Method Adaptation . . . . . . . . . . . . . 10

3.1 The Gaussian Radial Basis Function . . . . . . . . . . . . . . . . . 15
3.2 Analytical and Numerical Derivatives of Poisson’s Equation . . . . 21
3.3 Region of High Second Derivative Error in Poisson’s Equation . . . 22
3.4 Comparison of Derivative Methods Using Poisson’s Equation . . . . 22
3.5 Convection-Diffusion Equation Exact Solution and Approximation . . 25
3.6 Accuracy of Analytical and Numerical First Derivatives of the
   Convection-Diffusion Equation . . . . . . . . . . . . . . . . . . . . . 26
3.7 Accuracy of Analytical and Numerical Second Derivatives of the
   Convection-Diffusion Equation . . . . . . . . . . . . . . . . . . . . . 27
3.8 Error Over the Domain of Convection-Diffusion Equation . . . . . . 27
3.9 Convergence of the Convection-Diffusion Equation . . . . . . . . . . 28
3.10 ONERA M6 Nodal Distribution . . . . . . . . . . . . . . . . . . . . . 30
3.11 ONERA M6 $C_p$ Plot . . . . . . . . . . . . . . . . . . . . . . . . . . 31

4.1 Adding Nodes to the Domain . . . . . . . . . . . . . . . . . . . . . . 33
4.2 Assigning Approximation Values and Derivatives . . . . . . . . . . . 36

5.1 Adapted Convection-Diffusion Equation Using Basis Save Method . . 42
5.2 Error in Domain of Adapted Convection-Diffusion Equation Using
   the Basis Save Method . . . . . . . . . . . . . . . . . . . . . . . . . . 43
5.3 Convergence of Adapted Convection-Diffusion Equation Using the Basis Save Method ............................................. 44
5.4 Extended Error Progression of Adapted Convection-Diffusion Equation ................................................................. 45
5.5 Adapted Poisson’s Equation Using the Basis Save Method ............................................................. 46
5.6 Error over the Domain of Adapted Poisson’s Equation Using the Basis Save Method .................................... 47
5.7 Error Progression over the Domain of Adapted Poisson’s Equation Using the Basis Save Method .................. 48
5.8 2D Sine Function Approximation Adapted Using the Basis Save Method ...................................................... 50
5.9 2D Sine Function Convergence Adapted With the Basis Save Method ......................................................... 51
5.10 Global vs. Local Error Using the Basis Save Method .............................................................. 53

6.1 Interpolation Strategy Applied to the Convection-Diffusion Equation ......................................................... 56
6.2 Convection-Diffusion Derivatives using Numerical Derivatives ............................................................... 57
6.3 Poisson’s Equation After Interpolation Adaptation ......................................................................................... 59
6.4 Error of Poisson’s Equation After Interpolation Adaptation ......................................................................... 60
6.5 Error Convergence of Poisson’s Equation After Interpolation Adaptation ......................................................... 61
6.6 Convection-Diffusion Equation Adapted Using Semi-Interpolation Method ....................................................... 62
6.7 Error of the Convection Diffusion Equation Adapted Using Semi-Interpolation ...................................................... 63
6.8 Error Convergence of the Convection Diffusion Equation Adapted Using Semi-Interpolation ......................... 64
6.9 Poisson’s Equation Adapted using Semi-Interpolation Method .......................................................................... 65
6.10 Error of Poisson’s Equation Adapted Using Semi-Interpolation Method ......................................................... 65
6.11 Error Convergence of Poisson’s Equation Using Semi-Interpolation Method ..................................................... 66
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Error over Domain of Convection-Diffusion Equation with Optimized</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>Coefficients</td>
<td></td>
</tr>
<tr>
<td>A.2</td>
<td>Error Convergence of Convection-Diffusion Equation with Optimized</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Coefficients</td>
<td></td>
</tr>
<tr>
<td>A.3</td>
<td>Error over Domain of Convection-Diffusion Equation with Calculated</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>Coefficients</td>
<td></td>
</tr>
<tr>
<td>A.4</td>
<td>Error Convergence of Convection-Diffusion Equation with Calculated</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>Coefficients</td>
<td></td>
</tr>
<tr>
<td>B.1</td>
<td>Poisson’s Equation Adapted Using the Gradient</td>
<td>86</td>
</tr>
<tr>
<td>B.2</td>
<td>Error of Poisson’s Equation Adapted Using the Gradient</td>
<td>86</td>
</tr>
<tr>
<td>B.3</td>
<td>Error Convergence of Gradient Adapted Poisson’s Equation</td>
<td>87</td>
</tr>
<tr>
<td>B.4</td>
<td>Initial Nodal Distribution using a Half Sparse Mesh on Poisson’s</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>Equation</td>
<td></td>
</tr>
<tr>
<td>B.5</td>
<td>Nodal Distribution using a Half Sparse Mesh after Adaptation on</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>Poisson’s Equation</td>
<td></td>
</tr>
<tr>
<td>B.6</td>
<td>Error over Domain of Adapted Sparse Poisson’s Equation</td>
<td>89</td>
</tr>
<tr>
<td>B.7</td>
<td>Error Convergence of Adapted Sparse Poisson’s Equation</td>
<td>90</td>
</tr>
<tr>
<td>C.1</td>
<td>Example Training and Test Grid</td>
<td>92</td>
</tr>
<tr>
<td>C.2</td>
<td>Adapted Training Grid</td>
<td>93</td>
</tr>
<tr>
<td>C.3</td>
<td>Adapted Test Grid</td>
<td>94</td>
</tr>
<tr>
<td>C.4</td>
<td>Error Convergence of the Training Grid</td>
<td>95</td>
</tr>
<tr>
<td>C.5</td>
<td>Error Convergence of the Test Grid</td>
<td>96</td>
</tr>
<tr>
<td>C.6</td>
<td>Fine Test Grid Approximation Demonstrating High Error</td>
<td>97</td>
</tr>
<tr>
<td>C.7</td>
<td>Fine Test Grid Error Plot</td>
<td>98</td>
</tr>
</tbody>
</table>
Nomenclature and Abbreviations

$\alpha$  Angle of attack

$\epsilon$  Scalar Error

$\mathbb{R}^d$  Problem domain

$\phi_k(x)$  Basis function

$c_k$  Coefficient parameter

$C_p$  Coefficient of pressure

$D$  Dimension of the approximation

$d$  Distance between original and additional nodes

$F(x)$  Exact solution

$f_N(x)$  SOMA’s approximation

$f_o(x)$  Initial approximation

$g$  Forcing function

$H[\cdot]$  Function operator

$M$  Number of nodes of the approximation

$M_\infty$  Free-stream Mach number

$N$  Number of basis functions
$p_{(1,2),k}$ Optimization parameter for coefficients

$r_k$ Approximation error

$R_N(x)$ Residual of approximation

$Re$ Reynold's number

$s(x)$ Boundary driver equation

$u_N(x)$ SOMA’s approximation

$w_k$ Width parameter

$x_a$ New nodes added to the approximation

$x_c$ Center parameter in $x$-direction

$y_c$ Center parameter in $y$-direction

CFD Computational Fluid Dynamics

GRBF Gaussian Radial Basis Function

MQRBF Multi-Quadric Radial Basis Function

PDE Partial Differential Equation

RBF Radial Basis Function

SOMA Sequentially Optimized Meshfree Approximation
Chapter 1

Introduction

Meshfree Computational Fluid Dynamics (CFD) techniques are a widely researched field for visualizing complex fluid flow and interaction [1–3]. Meshfree techniques solve many problems associated with typical finite element or difference methods, especially the requirement of explicit connectivity between nodes in the domain. The Sequentially Optimized Meshfree Approximation (SOMA), is one such method which does not require a user-defined mesh to solve complex high order differential equations [4]. This method uses basis function collocation, which allows for infinite derivatives and the use of the strong-form of equations of interest instead of Galerkin weak forms [5]. SOMA is a powerful tool for solving various differential equations, but it was only implemented on a fixed-grid before work for this thesis began.

Adaptive techniques have also become extremely important to CFD solvers in recent years and many different methods for adapting various grids have been developed. When tackling large scale problems, typically the more nodes there are in an approximation, the more accurate and reliable the results. Of course, the trade-off is the amount of time that the problem takes to compute, so infinitely dense grids are not possible by practical standards. Researchers have come up with various adaptive methods to increase grid density where it is needed in the problem domain in order to capture important flow characteristics while leaving areas that do not affect the solution untouched. These adaptive techniques seek the benefits of using a locally denser grid without increasing computational time. Researchers have developed three
main methods of adapting grids, the r, h, and p methods. The first method, relocation (r), seeks to redistribute existing nodes to areas where they are needed. The second method, grid refinement (h), adds nodes to or removes nodes from the domain in regions of interest. The third method, p, varies the polynomial order of the basis function. The adaptive techniques applied to SOMA in this thesis are a combination of r and h methods.

Methods used for moving nodes or adding nodes to a domain can become difficult when they are applied to finite element methods because of the sensitivity of element shape to the convergence to an appropriate solution. Collocation methods like SOMA solve some of these problems. SOMA requires no connectivity data, which means that nodes can be added to the domain without the concern of creating misshapen elements or spoiling the convergence of a solution. However, this does not mean that adding nodes within the domain of SOMA is trivial. Calculating the approximation values and derivatives of every new node added to the domain has proven to be the primary challenge of this work, and two methods are presented in this thesis to address this challenge.

The first part of this thesis gives a basic overview of adaptive methods and SOMA. Chapter 2 presents the developing and current state of adaptive techniques. Ideas from both finite elements and collocation methods were used in creating the adaptive method for this thesis, so both are discussed in Chapter 2, focusing on how the various methods have been applied and their possible applications to SOMA. Chapter 3 begins a discussion of how SOMA works as a differential equation solver. A full derivation and explanation of SOMA was written by Wilkinson [4]. For brevity and clarity, only the parts of SOMA which are critical to adaptation are discussed in great detail.

In Chapter 4, the method of adding nodes near areas of high residual is presented.
This technique is based on edge-bisection techniques found frequently in adaptive literature. The same method for adding nodes within the domain is used for all of the examples in this thesis. The real challenge of this work is determining good approximation values for the new nodes as well as the derivatives of each node.

The first method for doing this and the first examples of adaptive techniques applied to SOMA are presented in Chapter 5. This first method, the Basis Save method, is wide in its applications and very robust, although there are some problems associated with its use. Chapter 6 presents an alternative method of finding approximation locations and derivatives of new nodes in the domain. This interpolation method seeks to solve some of the problems associated with the Basis Save method. Included in Chapter 6 is a discussion of how the two methods can merge, which truly proves the flexibility of SOMA as an adaptive solver.

Chapter 7 presents some conclusions drawn from the research and some future work for SOMA. Finally, three appendices contain additional information and present other work conducted, further proving the robustness and flexibility of SOMA as an adaptive algorithm.
Chapter 2

Overview of Adaptive Techniques

Adaptive numerical techniques that can solve various differential equations have been an important research topic in recent years. Scientists have developed many different schemes for adaptive meshing, and there are similarities and differences between many of these schemes. Some are developed specifically for finite element or finite difference methods; others have been developed for collocation-type methods such as SOMA. The purpose of this chapter is to explore those various schemes and to point out the details used to create an adaptive technique for SOMA.

2.1 Adaptive Finite Element Type Methods

Much research has been conducted on adapting traditional flow solvers that use finite element, finite volume, or finite difference methods. These methods have gained so much traction in the past decades because of their extensive range of applications.

There are three general methods for adapting finite element type meshes, two of which are worth explaining in this section. The first method moves existing nodes within the domain so they concentrate at certain regions of interest, typically areas of rapid variation. This is referred to as r-refinement [6]. The second involves placing new fixed nodes in locations within the domain to enrich the mesh at these regions of interest. This method is called h-refinement [7]. The following subsections will explain both techniques.
2.1.1 Nodal Movement Techniques

Techniques that intelligently move nodes to locations where they are needed have many benefits and applications. These movement methods can be further divided based on their redistribution class, of which three will be discussed in this subsection. The first type of redistribution seeks to minimize or equidistribute the error measure. Denny et al. [8] point out that improved accuracy solving differential equations numerically can be achieved by minimizing local truncation error. They minimize the truncation error in various two-point boundary value problems by optimizing the location of the nodes by differentiating truncation error with respect to the nodal position. Gough et al. [9] suggest that although uniform meshes are commonly chosen where there is a slowly varying solution to a differential equation, this is not often the best way to analyze regions such as boundaries with high gradients. They propose a stretching technique that seeks to minimize their error measure by using a stretching transformation between the physical frame $X$ and a new frame $\xi$.

The second method of moving nodes involves attracting and repelling certain nodes towards and away from each other. Nodes with a high truncation error typically “attract” other nodes, pulling the mesh towards them, whereas nodes with a lower error “repel” others. Notable development of such a method comes from a series of papers written by Rai and Anderson [10–12]. Their 1982 paper develops an attractive-repulsive method [12]. This method applies a pseudo “gravitational force” to nodes to pull them closer or push them apart depending on their individual values of error and spacing. The total attractive and repulsive force that each node experiences causes the node to move with some mathematical “velocity.” Many methods use error as the driving variable to adapt the mesh, but any variable can be used in this flexible method [12]. By inducing a rotation to the grid, the grid lines can be aligned to
the geometry of regions of high gradients such as shocks [13]. Element collapse is a problem that is fixed by increasing the repulsion of adjacent nodes as they get closer. Mathematical damping is used in these methods to help keep the grid from oscillating. This attracting and repelling technique can be thought of as an iterative way to achieve the same goal as the equidistribution methods discussed in the previous paragraph.

The final nodal movement technique to be discussed is developed by Miller and Miller in a series of papers from 1981 [14, 15]. They develop a technique where the location of nodes becomes an additional dependent variable within the finite element formulation [13]. A Galerkin formulation uses the solution and the grid location as variables for adaptation. This makes it possible to obtain the nodal location as well as the solution simultaneously. Various mathematical techniques are implemented to keep nodes from moving too close to each other such as repulsive forces [15] or internodal viscosity.

Nodal movement techniques are extremely useful for solving various partial differential equations. One particular benefit is that movement techniques keep the same number of nodes within the mesh, which means that the solution time is the same for a fixed or adapted mesh [6]. As mentioned above, these movement techniques are not the only technique used to adapt meshes. The following subsection will discuss enrichment, or h-refinement, techniques.

2.1.2 Mesh Enrichment Techniques

Various mesh enrichment, or h-refinement, schemes have been developed for solving differential equations. With finite element methods, typically elements with high values of some monitor parameter are selected for further refinement. In their 1992
paper, Löhner et al. [16] list three main “ingredients” to refined grids. The first is an optimal mesh criterion, the second is an error metric to determine where to refine the mesh, and the third is a refinement scheme. The rest of the section will seek to answer how researchers have compiled these ingredients into various methods of adaptive refinement.

All refinement techniques seek out areas in the domain of interest that require additional nodes. To refine these chosen areas, nodes are typically added in a logical pattern. For example, if the problem is one-dimensional, new nodes can be added to bisect the line between the two original nodes. For higher dimensional problems, such as two-dimensional problems using triangle elements, edge bisection is used to create two to four new triangles from the original element. Quadrilaterals are treated the same way, creating four new elements [6]. Berger et al. [17] are some of the first to demonstrate this method with steady Euler equations applied to an airfoil in transonic flow. Their algorithm automatically seeks out areas with a high error estimation and applies a uniformly refined grid to those areas, which are typically leading and trailing edges. Their results greatly decrease computational time, without sacrificing convergence.

Another method that uses triangular elements is presented by Frey in his 1987 article [18]. He notes that Delauney triangulation can be used to create meshes with well-shaped triangles for the required node spacing. The user only needs to provide locations of a few boundary nodes, and his algorithm can automatically create an adapted mesh with well-shaped elements. More two-dimensional mesh research was conducted by Rausch et al [19]. Flow properties are calculated with h-refinement techniques using an unstructured, two-dimensional mesh. Their work on steady and unsteady transonic flow over an airfoil utilizes mesh enrichment and deletion near
regions of high and low gradients, respectively. Large computational savings are achieved with their methods while maintaining accuracy by comparing with experimental data. Similar techniques can be found in three-dimensional cases as well.

Löhner et al. present an h-refinement method for transient problems in three-dimensions using tetrahedral elements [16]. These tetrahedral elements require a set of refinement rules in order to keep them from becoming ill-conditioned. This refinement technique also includes deletion, or removal of nodes based on criteria similar to adding nodes. A numerical example of shock-shock and shock-structure interaction demonstrates that their method saves computational time as well as storage capacity. One year later, Kallinderis et al. use tetrahedral elements to further improve these refinement schemes in three-dimensions by solving the problems associated with interface nodes [20]. In their 1998 article, Biswas et al. solve a similar problem as Löhner and Kallinderis with hexahedral elements [21]. In finite element-type adaptations, tetrahedrals can become poorly shaped after many consecutive adaptations. Poorly shaped grids can have disparities including element size or edge angles, and, by definition, poorly shaped grids always lead to inaccurate flow solutions [22].

Mesh refinement techniques have been widely studied and applied in finite element analyses. In fact, the method used to adapt SOMA presented in this thesis is a refinement technique. Unlike the adaptation schemes previously discussed, SOMA is not a finite element or difference method, but rather a collocation method. SOMA and other collocation techniques have some important characteristics that can mean fewer problems when adapting. The following section will discuss these collocation methods and the way researchers in the past have adapted them.
2.2 Adaptive Collocation Methods

Collocation methods such as Radial Basis Function (RBF) collocation have received considerable amounts of attention in recent years, and have become a viable option for solving various numerical problems. Some differences in these methods compared to finite element or difference methods make them a prime candidate for applying adaptive techniques. For example, collocation methods give a continuous, analytical solution for differential equations [23]. They also typically provide a higher order of convergence and are simple to implement. Because of the many attractive features of RBF methods, much effort has been made to integrate adaptive techniques [24].

Most RBF adaptive methods work in a similar manner. Typically, nodes are selected within the approximation that meet some sort of criteria, such as a value of error [25]. Nodes are then added or subtracted depending on the criteria, adapting the grid to increase its efficiency. Some algorithms also work using nodal movement, similar to the techniques discussed in Subsection 2.1.1. The algorithm then continues with the new nodal distribution, and some number of iterations later is again adapted. This process is repeated until some fitness metric is met. Figure 2.1 is a generalized flowchart showing the typical progression of an adaptive algorithm using RBF collocation methods [26].

Various adaptive RBF schemes have been developed, and the rest of this section will discuss some of problems that have been solved as well as the general form of these techniques. One of the first papers to appear that seeks to adapt a collocation-type scheme comes from Huang et al. [23]. They create a nodal movement method that seeks out areas of rapid solution variation. Their technique simultaneously solves the solution PDE as well as a grid discretization taken from finite difference methods. Theirs is the first “moving-mesh” collocation technique, and the results demonstrate
Figure 2.1: Typical Flowchart for RBF Method Adaptation.

a much faster convergence rate as well as higher accuracy than typical moving finite element methods. However, one pitfall of their method is that it is only really useful in small or medium sized grids.

Larger scale problems are tackled using an adaptive, greedy algorithm presented by Hon et al. [27]. By treating these large problems as a small problem, computational overhead can be greatly decreased without sacrificing accuracy. Their method uses a subset of nodes taken from a fine grid to progress the approximation, which decreases computational overhead. The rest of the nodes can then be used for error checking since the full analytical approximation over the domain is known. This method shares similarities with the Basis Save method presented in Chapter 5.

A training method was studied by Chen in a series of papers. Chen et al. [26, 28]
present an adaptive technique applied to a multi-quadric RBF (MQRBF) scheme using a residual-based training method. Training methods use a sparse grid and RBF network to produce accurate results that can subsequently be applied to a much finer grid. These training methods seek to improve the efficiency of the approximation by only requiring optimization calculations on the sparse grid, while hiding the fine grid until the sparse one is sufficiently adapted [29].

Similar techniques were applied to the nonlinear Schrödinger equation by Sanz-Serna and Christie [30], proving how adaptive RBF techniques can be used in a wide variety of problems. In his 2005 article, Sarra demonstrates how using the actual basis function equations, instead of interpolation steps, can be used to maintain high orders of accuracy, especially near the boundaries [1]. Sarra implements the adaptive scheme on Burger’s equation, proving his method works for equations that step in time. Conclusions from this study point to the benefits of adapting RBF methods in terms of accuracy and ease of implementation.

RBF networks are great candidates for adaptation for a multitude of reasons. It is logical, then, that SOMA should be considered an appropriate candidate for applying adaptive schemes. The following chapter will describe SOMA and its inner workings especially with regards to adaptation. A more comprehensive explanation of SOMA and its various applications was written by Wilkinson [4].
Chapter 3

The Sequentially Optimized Meshfree Approximation

SOMA is a meshfree algorithm that is capable of solving various types of problems including simple function approximation, linear partial differential equations (PDEs), and nonlinear PDEs [4,31]. Previous work on SOMA and its components have shown it to be a robust CFD solver [4,31]. SOMA is a suitable solver for adaptation because it does not require difficult meshing data. This lack of meshing data makes the approximation flexible, and it is a great jumping-off point for future research and development. This chapter provides information about SOMA as a tool to solve partial differential equations, focusing on specific characteristics which are important to making the algorithm adaptive. To describe how SOMA works, we begin by constructing its framework: an initial approximation with the addition of radial basis functions.

3.1 The Approximation

SOMA begins with an initial approximation that satisfies required boundary conditions for the particular problem. To this initial condition, $N$ basis functions are added in a linear expansion of the approximation. Addition continues until some metric has been met, typically a satisfactorily reduced residual or a user-defined maximum $N$. 
The approximation has the following form

\[ F(x) \approx f_N(x) = f_o(x) + \sum_{k=1}^{N} c_k \phi_k(x), \quad (3.1) \]

where \( F(x) \) is the exact solution of the problem at hand, \( f_N(x) \) is the value of the approximation after \( N \) basis functions have been added, and \( f_o \) is the initial approximation that satisfies the Dirichlet conditions. The parameter \( c_k \) is a linear coefficient that determines the height of each basis function. Each basis function is represented by \( \phi_k \), and will be further explained in Section 3.2. This approximation creates a continuous, and analytically differentiable equation to approximate a problem solution. This linear approximation that SOMA compiles will become very important for advancing the algorithm to adaptivity. To further describe SOMA and the way the approximation is adapted, additional information about its use of radial basis functions is necessary.

### 3.2 Radial Basis Functions

Development of RBF techniques spans almost half a century, and their applications range from solving PDEs to meteorology and robotics [32, 33]. RBF techniques have several advantages over finite element or finite difference methods, most notably the lack of a mesh with its associated connectivity data. Because RBFs can be applied to sets of nodes arbitrarily dispersed within a problem domain, computational cost is greatly reduced without the need to worry about element geometries [3].

RBF approximations, such as SOMA, are typically constructed using the following form:

\[ f_N(x) = \sum_{k=0}^{N} c_k \phi(\vec{x}). \quad (3.2) \]
The approximation is defined by $f_N(x)$, and the coefficient, $c_k$, is calculated within the algorithm. The parameter $\phi(x)$ is a chosen function defined over the domain, $\mathbb{R}^d$.

There exist many different functions used as $\phi(x)$ [24, 34], and they basically fall into two categories: infinitely smooth and differentiable or piecewise-smooth [35]. The basis functions used in SOMA and this thesis, Gaussian Radial Basis Functions (GRBFs), are an example of an infinitely-smooth, differentiable basis function. This type of basis function contains shape parameters that can be chosen and subsequently stored within the algorithm [1]. Because GRBFs are infinitely differentiable, high order differential equations can be solved using SOMA, while maintaining exponential convergence [4]. Figure 3.1 is a picture of a generic GRBF centered at the origin, with an uncharacteristically large coefficient (or height) for visualization purposes. GRBFs take the following form in one-dimension:

$$\phi_k(x) = e^{(w_k \|x-x_{c,k}\|^2)}, \quad (3.3)$$

where $\vec{x}$ is a given vector located within the problem domain, $\mathbb{R}^d$, and the Euclidean distance between two nodes is shown as $\| \cdot \|$ [4, 26, 32, 35]. Here, $w_k$ is a “width” parameter that determines the steepness of the basis function. As $w_k$ becomes more negative, the GRBF $\phi_k(x)$ approaches $\delta$; conversely as $w_k$ approaches zero, $\phi_k(x)$ will become a non-zero constant. The last parameter in Equation 3.3, $x_{c,k}$, determines the center location of the GRBF and will be referenced as the “center” parameter in this thesis. SOMA does not add GRBFs to the initial approximation haphazardly, but instead seeks out a set of parameters that reduce the error within the approximation.

Optimization of the width, $w_k$, and center, $x_{c,k}$, takes place within the optimization routine in SOMA. The linear coefficient is discussed in Section 3.4 and Appendix A. Within the genetic algorithm used in this work, the width parameter, $w_k$ is unbounded, but the initial range of values possible for the width ranges from 100 to $-100$. The
center parameter has stricter bounds; it is initialized and bounded within the domain of the problem, typically $0 \leq x \leq 1$ for one-dimensional problems. The genetic algorithm used is flexible, and these bounds can be altered to try to affect convergence, however optimizing the ranges of the parameters is beyond the scope of this thesis.

### 3.3 Residual Calculations

Within each iteration, SOMA seeks to minimize the equation residual, and each basis function is optimized to meet this requirement. The residual at each individual node is a known value, and so to keep track of the overall accuracy of the approximation, a merit function is used to obtain a scalar value of the residual. For any given basis
function, \( N \), the residual of the approximation is given by

\[
R_N(x) = H[c_N\phi_N + f_{N-1}] - g,
\]

(3.4)

where \( H[\cdot] \) is some function operator specific to the problem, which includes the boundary conditions, \( g \) is a forcing function, and \( f_{N-1} \) is the compiled approximation at \( N - 1 \) basis functions. The overall scalar error of the approximation is useful because it allows the user to follow the progression of the approximation without getting lost in the details of the error at each individual node. The scalar error can be calculated using the following:

\[
\epsilon = \langle R_N, R_N \rangle,
\]

(3.5)

where \( \langle \cdot, \cdot \rangle \) is the inner product, evaluated at the nodes [4, 36]. Because the number of nodes changes whenever the adaptive algorithm is implemented, it does not make sense to compare the error of the original approximation to the error with the increased number of nodes. Therefore, the scalar value of the residual will typically take the form

\[
\epsilon = \frac{1}{M} \langle R_N, R_N \rangle,
\]

(3.6)

where \( M \) is the number of nodes within the approximation. Normalization of the error using the number of nodes in the approximation is a typical metric for error for RBF methods [1,24]. Many different properties of SOMA can affect how well the algorithm reduces the error of the approximation. Two of the major contributors to the accuracy and efficiency of the approximation are the method in which the linear parameter, \( c_k \), is calculated and the enforcement of boundary conditions. The following sections will discuss the way that \( c_k \) is determined in SOMA and how boundary conditions are enforced.
3.4 Determining the Coefficients in SOMA

Determining the linear coefficient, \( c_k \), for each individual basis function has proved to be a challenge in SOMA. Traditionally, for a linear differential operator \( H[\cdot] \), the value of the coefficient is calculated using the following equation:

\[
c_k = -\frac{\langle H[\phi_k], R_{N-1} \rangle}{\langle H[\phi_k], H[\phi_k] \rangle},
\]

where \( R_{N-1} \) is the form of the residual after \( N - 1 \) basis functions and \( H[\phi_k] \) is the equation operator on the \( k^{th} \) basis function. Wilkinson [4] gives a full derivation of this equation.

Through some experimentation with SOMA while trying to make the algorithm more efficient, it was discovered that calculating the coefficient in this manner may not necessarily be the best method. Alternatively, the coefficient can be a parameter included in the same optimization routine that calculates the value of the width coefficient, \( w_k \), and the center, \( x_{c,k} \). This is a very recent discovery within SOMA, and more information about optimized coefficients can be found in Appendix A.

The final step towards constructing each basis function is enforcing the boundary conditions inherent to the differential equation SOMA is trying to solve. The following section will explain one of many ways that SOMA enforces boundary conditions.

3.5 Enforcing Boundary Conditions

SOMA can enforce boundary conditions for individual differential equations in different ways. The boundary conditions explored in this thesis are all Dirichlet conditions, which require the value of the approximation be fixed for every iteration. The option chosen for work in this thesis is a boundary driver equation which alters the shape of each basis function so the boundary conditions are satisfied.
Each basis function is multiplied by the boundary driver equation, which forces the value of the basis function to be zero at the boundaries. If the initial condition given by the user satisfies the boundaries conditions, then each subsequent iteration will also satisfy them because no basis function will change the value of the approximation at the boundary. Equation 3.8 demonstrates the boundary driver equation used for one-dimensional problems:

$$s(x) = (x - x(LB))(x(UB) - x), \tag{3.8}$$

where $s(x)$ is the boundary driver equation, and $x(LB)$ and $x(UB)$ are the lower and upper boundaries of the domain of interest. So, for example, if the domain of interest is $0 \leq x \leq 1$, the boundary driver equation will be:

$$s(x) = (x - x^2). \tag{3.9}$$

This method enforces the boundary conditions at each iteration, often times at the expense of the equation residual especially near the boundaries. To demonstrate how the boundary drivers affect SOMA’s approximation, the derivatives of the approximation must be discussed.

### 3.6 Calculating Derivatives in SOMA

When approximating differential equations, derivatives of the solution of the operator, $H[\cdot]$, determine how well the differential equation is satisfied. It is well documented that accurate derivatives are imperative to an accurate and efficient convergence of a solution [31]. In preparation of this thesis, both analytical and numerical techniques for calculating the derivatives of the approximation were tested.

Because the method of calculating the value of the approximation and the value of the derivatives of that approximation do not always coincide, it will be necessary to
fully explain how the derivatives are calculated for each example throughout this thesis. In some cases, the derivatives may be calculated using the Basis Saving method outlined in Chapter 5. These derivatives represent the sum of the derivatives of all the basis functions added to the approximation. This type of derivative is typically referred to as the “analytical derivative” of the approximation. Other cases may use a “numerical derivative” method, which has the advantage of being fast, although it is not accurate to machine precision [36]. The numerical technique used throughout this thesis is one based on a second order polynomial fit at each sequence of three consecutive nodes. Weights are assigned to each node depending on node spacing [37]. Because the values of derivatives are critical to efficiently adapting the approximation, the following subsection will demonstrate the accuracy of the two different methods using Poisson’s equation as an example. Subsection 3.6.2 will discuss some difficulties calculating derivatives in SOMA using the convection-diffusion equation as an example.

Throughout this thesis, the convection-diffusion equation and Poisson’s equation will be used as test cases for the adaptive methods presented. Using more than one equation to demonstrate adaptive methods prevents errors from perpetuating and helps to prove the relevance of adaptive techniques.

The convection-diffusion equation represents a pared-down version of the full Navier-Stokes equations. Along with an accumulation and source term, the full Navier-Stokes equation also includes a convective and a diffusive term. This makes the convection-diffusion equation a good example for developing robust adaptive techniques. Another practical benefit of using the convection-diffusion equation is that it contains both first and second order derivatives, which make it an ideal equation for a comprehensive algorithm.
Poisson’s equation is an elliptic differential equation and is used in a wide variety of applications including electrostatics, physics, and engineering. The boundaries of Poisson’s equation may be Dirichlet, Neumann, or a combination of the two. It is frequently used for testing due to its wide range of applications and its relative simplicity compared to other elliptic PDEs.

### 3.6.1 Derivatives and Poisson’s Equation

The one-dimensional, linear solution to Poisson’s equation has exact derivatives that are easily computed and is therefore a good benchmark to demonstrate how derivatives can be calculated within SOMA. Poisson’s equation is written as follows:

\[ \nabla^2 F - g = 0, \]  

(3.10)

where \( F \) and \( g \) are real functions that satisfy the equation. More than one function \( F \) and \( g \) can be chosen to satisfy the equation. The solution chosen for this thesis is

\[ F = \frac{1}{12} (x - x^4). \]  

(3.11)

It follows, then, that the second derivative of \( F \), \( g \) is

\[ \nabla^2 F = g = -x^2. \]  

(3.12)

The Dirichlet boundary conditions that must be satisfied for each iteration are \( F(0) = 0 \) and \( F(1) = 0 \) over the domain \([0, 1]\). Poisson’s equation requires the use of only the original function, \( F \), and its second derivative, \( g \), so its first derivative may be ignored. The second derivative in Equation 3.12 is considered the “exact” second derivative of the approximation.

It is important to actually determine how well SOMA approximates the derivatives of an equation such as the Poisson’s equation. Figure 3.2 is a plot that shows how close
both analytical and numerical derivatives are to the exact derivative. The absolute value of the derivatives are displayed for ease of comparison. In fact, the derivatives are so close that a zoomed-in figure seems necessary to really show the difference. The majority of the error is located at the boundaries where SOMA typically has a more difficult job resolving the approximation. Figure 3.3 shows an area of high derivative error in the approximation using the two schemes for calculating the approximation derivative and the exact derivative. It is important to notice the scale on the y-axis of all of the figures. The orders of magnitude of the difference between the analytical and numerical derivatives of the approximation are very small. So small, in fact, that it is also useful to determine actual differences between the analytical and numerical derivatives. The difference between the methods will let us know if there is a distinct advantage to using either the analytical or the numerical derivatives. Figure 3.4 demonstrates the difference between analytical and numerical derivatives of the

Figure 3.2: Plot comparing numerical and analytical second derivative of the approximation of one-dimensional, linear Poisson’s equation compared to the exact solution’s second derivative. Note how closely the numerical and analytical derivatives match with the exact second derivative.
Figure 3.3: Zoomed-in plot showing an area of high error in the second derivative of the approximation of one-dimensional, linear Poisson’s equation. Note the y-axis scaling: the difference between analytical and numerical derivatives is very small.

Figure 3.4: Semi-logarithmic plot of the absolute value of the difference between the numerical and analytical derivatives of the one-dimensional, linear Poisson’s equation. The y-axis is on a logarithmic scale and demonstrates the small difference between analytical and numerical derivatives.
numerical derivatives compare. The left side of the plot where \( x < 0.3 \) demonstrates a larger difference in the accuracy of these two methods, but the difference is still very small. Therefore it seems safe to use both methods interchangeably in this thesis as long as the same method is consistently used for every iteration and all orders of derivatives calculated. Numerical derivatives are much faster to calculate especially as problems advance in dimension and complexity, so in some cases these are certainly desired. However, no matter which of the two ways derivatives of the approximation are calculated, there are limitations inherent to SOMA which make the derivatives and therefore the error difficult to resolve.

3.6.2 Properties of Derivatives and Convergence in SOMA

It has been shown that the “map” of the analytical derivative to the numerical derivative over the set of nodes used in the approximation can become disconnected [5]. This disconnect stems from the derivatives of the basis functions added to the approximation. Inevitably, the value of a linear combination of the derivatives of the basis functions does not coincide with the numerical derivative of the approximation.

SOMA has a difficult time resolving derivatives near the boundaries of the domain of the approximation because boundary conditions are strictly enforced on every basis function at the expense of the equation residual. The error plots of the differential equations presented in this section as well as in the following chapters will demonstrate high values of equation residual near the boundaries. This high error is considered an artifact of the numerical techniques used in SOMA, and not a problem with the adaptive techniques presented in this thesis.

Boundary enforcement and convergence rates in SOMA are demonstrated by using the one-dimensional convection-diffusion equation. The boundary error for this
The problem is consistently large at both boundaries. The one-dimensional convection-diffusion equation is written as follows:

\[
\frac{d^2 F}{dx^2} - Re \frac{dF}{dx} = 0,
\]

(3.13)

where \( f \) is some equation that satisfies the differential equation, and \( Re \) is a user-defined Reynolds Number. For this problem, and unless otherwise stated, \( Re = 20 \).

The function, \( f \), which satisfies Equation 3.13 is written as

\[
F = \frac{e^{xRe} - 1}{e^{Re} - 1}.
\]

(3.14)

The first derivative of \( f \) is therefore

\[
\frac{dF}{dx} = \frac{Re(e^{xRe})}{e^{Re} - 1},
\]

(3.15)

and the second derivative is

\[
\frac{d^2 F}{dx^2} = \frac{Re^2(e^{xRe})}{e^{Re} - 1}.
\]

(3.16)

The only boundary conditions for this problem are Dirichlet boundary conditions, \( F(0) = 0 \) and \( F(1) = 1 \). This one-dimensional, linear solution to the convection-diffusion equation demonstrates one of the weaknesses of SOMA as a differential equation solver. In regions near to the boundaries, the approximation error can be well within an acceptable tolerance, but the equation residual, the right-hand-side of Equation 3.13, may be relatively high. This high equation residual of the approximation keeps the \( L_2 \) norm from converging quickly to an acceptable tolerance.

Figure 3.5 is a plot of the solution of the convection-diffusion equation demonstrated in Equation 3.14 after 100 basis functions in SOMA. Notice that the exact solution and SOMA’s approximation are constant \( x = 0 \) to \( x \approx 0.8 \). The approximation follows the curve closely up to its boundary value of \( y = 1 \). However, near the
Figure 3.5: Approximation and exact solution of the convection-diffusion equation after 100 basis functions. Note how closely the approximation and the exact solution seem to coincide. This represents a very low approximation error, but does not always coincide with a low equation residual.

boundaries, SOMA has a difficult time resolving the derivatives of the approximation. This difficulty is demonstrated using both analytical and numerical techniques to calculate the derivatives of the approximation. Figure 3.6 plots the difference between the exact first derivative of the convection-diffusion equation and the first derivative of the approximation after 100 basis functions using numerical and analytical techniques. In other words, this plot represents the difference between the calculated derivatives of the approximation and Equation 3.15.

Of course because this is a logarithmic plot, the absolute values of the derivatives are used so that no negative data is ignored. Areas where SOMA does a good job approximating the derivative are closer to zero, and typically these areas are near the center of the figure. The plot shows how the derivative seems to derail itself above $x > 0.9$. Similar results are demonstrated using the second derivative of the approxi-
Figure 3.6: Semi-logarithmic plot of the absolute value of the difference between first derivatives of the convection-diffusion equation and the exact first derivative after 100 basis functions. The numerical and analytical first derivatives seem to derail themselves near the boundaries.

Figure 3.7 shows the analytical and numerical second derivatives of the approximation after 100 basis functions. SOMA does a relatively good job near the center of the domain of interest. Near the boundaries, however, the error within the approximation can be very high and keep the scalar value of error from converging to an acceptable value. Figure 3.8 demonstrates how the boundary behavior shown in Figures 3.6 and 3.7 can affect the error of the approximation. The equation residual is shown as green circles and the approximation error is shown as blue triangles. The equation residual is typically the more difficult metric to minimize because it relies directly on the derivatives of the approximation, which are sacrificed near the boundaries where SOMA satisfies the boundary conditions. From Figure 3.8, it is clear that the equation residual is most accurate within $0.4 < x < 0.7$. Near the boundaries, the equation residual becomes relatively large.

As basis functions are added to the approximation, the scalar error values are
Figure 3.7: Semi-logarithmic plot of the difference between second derivatives of the convection-diffusion equation and the exact second derivative after 100 basis functions. The derivatives are nearest to the exact solution near the center of the plot, and farthest near the boundaries.

Figure 3.8: Plot of the approximation error and equation residual of the convection-diffusion equation after 100 basis functions. The equation residual is shown as green circles, and the approximation error is shown as blue triangles. Note how the equation residual is smallest in the center of the plot, away from the boundaries, and highest near the boundaries.
tracked to show how inaccurate derivatives can slow convergence of the approximation. Figure 3.9 demonstrates the scalar error convergence after 100 basis functions have been added to the fixed grid approximation. The green circles represent the equation residual, and the blue triangles represent the approximation error. A black line is labeled to demonstrate linear convergence. Figure 3.9 should be noted as a reference for future convergence plots. Note how the equation residual, shown in green, converges almost quadratically until approximately 20 basis functions and then slows to sub-linear convergence. Nearly linear convergence of the equation residual is considered the baseline performance that all adaptive plots will be compared to. The approximation error, shown as blue triangles, demonstrates quadratic convergence. Convergence plots in Chapters 5 and 6 demonstrate at least linear convergence of the equation residual and quadratic convergence for the approximation error, demon-
strating how SOMA can be improved with adaptation.

The scalar value of both the equation residual and approximation error is calculated for Figure 3.9 from the following normalized vector norm [1]:

$$
\epsilon = \frac{1}{M} \left( \sum_{k=1}^{M} |r_k|^2 \right)^{\frac{1}{2}},
$$

(3.17)

where $\epsilon$ is the normalized error value of the approximation, $r_k$ is the error vector across the domain (plotted in Figure 3.8) and can represent either the equation residual or the approximation error. The variable $M$ is the number of nodes within the domain of interest. The inclusion of $\frac{1}{M}$ is what makes this scalar error value “normalized,” and will be used throughout this thesis because it offers a more accurate comparison of scalar error values once extra nodes have been added to the domain. Because these extra nodes will of course increase the overall scalar error, normalizing with respect to the number of nodes is used to keep the scalar error consistent [1].

### 3.7 SOMA Applied to a Compressible Flow Problem

SOMA has been applied to various geometries and flow conditions, but one inviscid, compressible flow example stands out as a prime candidate for adaptation. The ONERA M6 wing has both taper and sweep, and therefore in transonic conditions it will develop a $\lambda$-shaped shock-shock interaction [4]. Adaptive schemes are especially useful for capturing flow structures such as shock waves, and the ONERA example will demonstrate how a fixed grid works well, but an adapted one may provide computational savings.

Figure 3.10 shows the nodal distribution used on the ONERA M6 wing. This distribution represents a mesh that was implemented using unstructured finite volume methods for comparison. The $(x, y, z)$ coordinates were fed into SOMA as simple list
of nodes. Note how the mesh, while unstructured, is very uniformly distributed.

Figure 3.10: Nodal distribution over the ONERA M6 wing for comparing SOMA to finite volume methods. This false mesh consists of 72,791 nodes [4].

The value of the pressure coefficient, $C_p$, is calculated over the wing using SOMA as well as finite volume methods for $\alpha = 3.06 \text{ deg}$ and $M_\infty = 0.8358$. The results of the comparison is shown in Figure 3.11. The finite volume results are shown on the left and SOMA’s results on the right. Visual inspection shows clearly the shock-shock interaction in the pressure coefficient isolines.

This ONERA M6 wing in transonic flow conditions would be a prime candidate for adaptation. While SOMA clearly approximates the solution well with a fixed grid, the benefits of an adaptive grid would be numerous. More nodes could be added near the locations of the $\lambda$-shock interaction. Fewer nodes could be included where the flow does not develop defined structures, such as the green area behind the $\lambda$-shock. Adaptation could also benefit the development of the flow field surrounding the wing, which is not shown in the previous figures. Areas near the wake, especially where
wing-tip vortices appear, would benefit from a refined mesh. Near the boundaries of the domain, where the flow approaches free-stream parameters, fewer nodes would be required.

Now that the basics of SOMA and its properties that are particularly important for this thesis are fully explained, it is possible to begin describing how to implement an adaptive algorithm. The following chapters describe how to add nodes within the domain, and how to incorporate these nodes into the approximation.
Chapter 4

Adaptive SOMA

When it comes to adaptivity, the most important feature of SOMA is the lack of explicit connectivity data between the nodes. With collocation methods such as SOMA and other RBF methods, nodes can be added and subtracted from the problem domain without any concern for ruining a well-spaced grid. Finite element and difference methods require a large amount of user-interaction and computational overhead to adapt the mesh [7]. With these methods, each node must be coordinated with other nodes so that they may be moved relatively evenly. Because SOMA is required to only minimize the error at specific nodes of interest, and not necessarily the entire domain as a whole, the user is free to adapt the current mesh as he or she wishes. As mentioned in Chapter 2, there are many possible methods for adapting collocation type methods. The adaptive techniques presented in this thesis are based on mesh refinement by adding new nodes to the domain.

4.1 Adding Nodes to the Domain

All of the methods presented in this thesis use mesh refinement near areas of high residual to construct a new vector of nodes, $x_a$, to take the place of the original domain, $x$. When the algorithm locates these areas, nodes are added a certain distance away from the original. In one-dimensional cases, the $x$-value of the new nodes is determined by placing them equally spaced to the left and right of the original node.
In two-dimensional cases, the nodes are added in a cross pattern surrounding the original node, similar to the one-dimensional cases. New nodes can be added at any distance away from the original, but typically the value is set to half the distance to the adjacent node unless otherwise specified [6].

Figure 4.1 : Scheme for adding nodes to areas of high residual in two-dimensions. The blue and yellow circles represent nodes that are part of the original grid. The yellow node represents a node with high error. The red circles represent new nodes added around that node of high error at some distance, $d$.

Figure 4.1 demonstrates the basic method of adding nodes to an approximation using SOMA. The $3 \times 3$ grid of eight light blue nodes and one yellow node represent original “parent” nodes in the $x$-$y$ direction before any adaptation algorithm has been implemented. SOMA seeks out nodes that meet or exceed some an error tolerance, represented by the yellow node in Figure 4.1. New nodes, shown in red, are then added to the approximation in a cross-pattern (or in only one direction if the problem is one-dimensional), at some distance, $d$, away from its parent node. These new nodes make up the previously mentioned $x_a$ vector of nodes used in the collocation. If the node of high residual lies on a boundary, then no nodes are added outside of that boundary in order to keep the domain consistent throughout the approximation. When the adaptive step is complete, and SOMA continues operating on the approximation,
the newly added nodes are treated identically to their parent nodes. For subsequent adaptations of the same problem, every node, no matter when it was added to the domain, becomes a parent node for the newest adaptation. This technique for locating new nodes within the domain is always implemented the same way even though there is more than one property besides the equation residual that can drive the adaptation. This gives us a consistent comparison of different background methods of determining the values of the new nodes added to the approximation.

### 4.2 Metrics of Adaptation

It is critical to mention the different possible metrics for determining where the approximation needs refinement. The adaptation of function approximation problems is driven by the approximation error at each individual node, and for the differential equations presented in this thesis, the equation residual at each individual node drives the adaptation. Other metrics demonstrated are the gradient or sparseness of the approximation. In these cases, the mesh is refined in areas where the approximation meets some sort of gradient criteria or when the distance between neighboring nodes is greater than some user-defined quantity.

In this thesis, nodes will be added to areas of high residual for a few reasons. This seems to be a typical standard for other collocation-type adaptive grids [3, 25, 28], although it is not the only parameter used in the literature [24]. The equation residual is particularly attractive for SOMA because it is a quantity that is calculated at every iteration no matter which equation SOMA is trying to solve. Some differential equations do not require the use of the first derivative, notably Poisson’s equation, which means that using the first derivative as a metric of adaptation would require an extra calculation not in the current scheme, possibly slowing down computations. Finally,
areas of high equation residual are typically those areas where further adaptation would be required with a gradient-type method, so the results are not too different.

Gradient and other methods are viable with the adaptive techniques presented in this thesis, but will not be explored in the main chapters. Results of gradient and sparseness-type adaptive methods did not prove to be particularly different than residual-based methods, and only prove that the adaptive methods presented are robust and flexible. Appendix B contains some results of adaptation using gradient and sparseness as the metrics of adaptation.

No matter which metric used, a tolerance must be set to determine where the regions of interest lie within the domain. A residual tolerance is determined for each experiment typically by knowing the approximate order of magnitude of the majority of the approximation error present. Increasing the tolerance to capture only a small portion of the approximation keeps from overloading the approximation. Occasionally tolerance changes within the algorithm, especially if no nodes in the domain exceed the original tolerance, or if the tolerance as-is would capture too much of the approximation [3]. Determining where the regions in need of adaptation are located within the domain is a relatively simple task compared to the bigger challenge of assigning approximation values to these new nodes.

### 4.3 Assigning Approximation Values to New Nodes

The techniques presented in previous sections to add nodes to the new vector, \( x_a \), of nodes in the domain can be applied to SOMA no matter what actual method of adaptation is used. The individual methods differ in the way that they place nodes in the \( z \)-direction, that is, the direction of the approximation. Nodes can be added anywhere in the domain using SOMA, but their location within the approximation
is a more sensitive parameter. Poorly placed nodes in the approximation direction can miss important information from the residual and ruin the overall error of the approximation, which causes SOMA to waste computational time. Figure 4.2 shows a slice in the $z$-direction of a general approximation similar to the one shown in Figure 4.1. Once again, the blue and yellow nodes represent the parent nodes in the approximation. The yellow node represents a node with a residual that exceeds the set tolerance. The red nodes are the new nodes which have been added to the approximation. From Figure 4.2, it can be easily seen how the placement of new nodes in the $z$-direction is critical to creating an adapted approximation that is at least as accurate as the approximation before adaptation. Even when including a normalizing factor, $\frac{1}{P}$, if the newly added nodes do not fit within the approximation, the total scalar error (see Equations 3.5 and 3.6) will increase significantly, possibly slowing down convergence. The following chapters provide solutions to the problem of assigning approximation values to the new nodes without ruining the overall approximation. Two basic methods for adding nodes to the approximation have been developed for SOMA. The first method, outlined in Chapter 5, is the Basis Save method, which saves each basis function for reconstruction. The second method, the Interpolation
method, solves some problems associated with the Basis Save method and is discussed in Chapter 6.
Chapter 5

The Basis Save Method

SOMA as a fixed-grid method uses a simplified version of the approximation for each iteration. The equation presented in Equation 3.1 demonstrates the theoretical basis of SOMA, but in practice, a simpler equation is used to save memory and limit the number of parameters the algorithm holds on to. Instead of calculating the sum of all previous basis functions, SOMA stores the previous approximation and adds the $N^{th}$ basis function to the current approximation which is a sum of all previous basis functions. The current basis approximation at $N$ basis functions is written as

$$f_N(x) = f_{N-1}(x) + c_N \phi_N(x, w_N, x_{C,N}),$$  \hspace{1cm} (5.1)

where the left-hand side of the function, $f_N(x)$, is the approximation and the right-hand side of the equation is the approximation at $N - 1$ basis functions plus the current basis function.

This form of the approximation makes calculating the newest iteration computationally simple because there is no need to save vectors full of coefficients. However, because all of the previous basis functions are compiled into $f_N$, there is no way to go back to look at individual basis functions. This becomes an important characteristic of the fixed-grid version of SOMA because saving each basis function has proven to be one of the best ways to add new nodes into the approximation.

The first method developed for adding nodes within the domain is referred to as the “Basis Save” method. The rest of this chapter will explain the method behind
the adaptation, construction of an adapted grid, as well as results of this method and some associated problems.

5.1 Method of Adaptation

As presented in Equation 5.1, each iteration is a combination of a new basis functions and the sum of all previous basis functions. Saving the parameters of each basis function adds an extra amount of memory usage to the algorithm, however until a very large number of basis functions are used, the coefficient matrix should not overload the memory of the machine. Chapter 6 presents a solution to the problems of parameter saving if the problem becomes very large. The matrix used to store these coefficients is equal to \([P \times N]\), where \(P\) is the number of parameters used for each basis function, and \(N\) is the number of basis functions. For the GRBFs used in this work, the number of parameters used in each GRBF is equal to the number of dimensions of the approximation plus one additional width parameter and one coefficient of the basis function. The value \(P\) for the GRBFs used to save each basis function in SOMA is therefore equal to \(D + 2\) where \(D\) is the dimension of the approximation [4].

The Basis Save method begins as SOMA traditionally does, by constructing an initial condition from which to start the approximation and intelligently adding a basis function to that initial condition. The approximation then takes the following form after that first basis function:

\[
f_1(x) = f_o(x) + c_1\phi_1(x, w_1, x_{C,1})
\]  

(5.2)

The parameters \(c_1, w_1,\) and \(x_{C,1}\) are then stored in a matrix, and the approximation is updated and advanced to the next basis function. This process is repeated and the
parameters are saved until the point at which the mesh is adapted. The number of basis functions that SOMA compiles before an adaptation scheme is implemented is a parameter chosen by the user, and typically for this work is between 10 and 50 basis functions. If SOMA is adapted before it has had a chance to develop a somewhat accurate approximation, there is a risk of over-populating the mesh. If every area of the domain contains very closely-placed nodes, the purpose of adapting is defeated.

When SOMA reaches the set adaptation point, the task of recompiling all of the basis functions begins. The initial condition that was used for the approximation is a known value, and is therefore used as the first input to the recompilation. By using the matrix full of saved coefficients, every basis function can be added as a function to the initial condition, recreating the approximation with an added benefit: nodes within the domain can now be changed. The new form of the approximation is the following:

$$f_N(x_a) = f_o(x_a) + \sum_{k=1}^{N} c_k \phi_k(x_a, w_k, x_{c,k}),$$

(5.3)

where $w_k$ is the width parameter, $c_k$ is the coefficient, and $x_{c,k}$ is the center location for the $k^{th}$ basis function. The variable $x_a$ represents the new vector of nodes within the domain after the adaptive algorithm described in Chapter 4 has been applied. Equation 5.3 represents the ever-important analytical approximation.

A further benefit of this reconstruction is the ability to calculate the analytical derivatives of the approximation.

### 5.2 Derivatives of Saved Basis Functions

Each GRBF added to the approximation represents an equation which is infinitely differentiable. A linear combination of these GRBFs, such as the one constructed in SOMA, is therefore also infinitely differentiable. If each basis function added to the
approximation has a known derivative, then the derivative of the approximation as a whole can be analytically calculated as a linear combination of the derivatives of the basis functions [34]. The derivatives of the basis functions used in one-dimensional differential equations include the boundary drivers, and are written as follows:

\[
\phi(x) = e^{w_N(x-x_{c,N})^2}(s),
\]

(5.4)

where \( w_N \) is the width parameter of the \( N^{th} \) basis function, \( x_{c,N} \) is the center value, and \( s \) is the boundary driver equation for that particular problem. The first derivative of 5.4 is:

\[
\phi_x(x) = e^{w_N(x-x_{c,N})^2}(s_x + s(2w_N(x - x_{c,N}))),
\]

(5.5)

where \( s_x \) is the first derivative of the boundary driver equation. The second derivative is then:

\[
\phi_{xx}(x) = e^{w_N(x-x_{c,N})^2}(s_{xx} + 2s_x(2w_N(x - x_{c,N})) + s(2w_N(x - x_{c,N}))^2 + s(2w_N)),
\]

(5.6)

where \( s_{xx} \) is the second derivative of the boundary driver. The boundary driver equations do not change within SOMA, and therefore do not need to be saved for each iteration decreasing the storage capacity required for adaptation. To reconstruct the entire approximation, including its derivatives, after some number of basis functions, not every parameter in the derivatives must be stored. Only the center and width parameters as well as the coefficient for the basis function.

The reconstructed equations can be manipulated however the user wishes. Specifically, any new set of nodes, \( x_a \), can be substituted for the original set from which the approximation was constructed. The new set of adapted nodes, \( x_a \) is used in place of the original \( x \) vector in both the analytical approximation and the analytical derivatives. These new nodes, deliberately placed, create the intelligently adapted mesh. The following sections present examples of the Basis Save method in action.
5.3 Adapting the Convection-Diffusion Equation

The one-dimensional convection-diffusion equation is the first example of using the Basis Save method with SOMA. The same equation used in 3.6.2 is used for this example, with $Re = 20$. The derivatives are calculated using the analytical derivatives of the approximation. After every 40 basis functions, the adding algorithm seeks out nodes of high equation residual, higher than a tolerance of $10^{-1}$ at 40 basis functions, and higher than $10^{-2}$ at 80 basis functions. The tolerance decreases because as the algorithm progresses there are fewer and fewer nodes with error above the original tolerance. This particular example is advanced to 120 basis functions. After 120 basis functions, the trends in the error can be easily seen while the nodal distribution is still sparse enough to see clearly the sparser and finer areas. Figure 5.1 shows the approximation in its final state after the 120 basis functions have been added. The

![Figure 5.1: The convection-diffusion equation after 120 basis functions, adapting every 40 basis functions using the Basis Save method. The exact solution is shown as a red line, and SOMA’s approximation is shown as blue circles. The nodal distribution is shown below the approximation as black circles.](image)
The red line denotes the exact solution to the convection-diffusion that SOMA is seeking. The blue circles represent the actual nodes of the approximation. Just from looking at the plot, it is easy to see that the approximation matches well with the exact solution. The line of black circles below represents the x-locations of the nodes. Nodes within the approximation are no longer evenly distributed, but are placed around areas of high equation residual. They seem to gather mostly around the boundaries of the approximation, which makes sense because SOMA has a difficult time accurately capturing the equation residual around boundaries.

The error of the approximation after 120 basis functions is shown in Figure 5.2. Green circles represent the equation residual of each node within the approximation after 120 basis functions. The blue triangles represent the approximation error after 120 basis functions. This figure demonstrates how SOMA often has a difficult
time resolving the derivatives around the boundaries. Nodes are added around areas where the absolute value of the equation residual is especially high, typically near the boundaries.

To really see how the adaptation algorithm affects SOMA it is useful to take a look at the error progression. Figure 5.3 shows the normalized scalar equation residual and approximation error for each basis function up to 120 basis functions. The approximation was adapted at 40 basis functions and again at 80 basis functions. The equation residual is the more important value to consider in this case because it is the parameter that SOMA chases with each basis function and is the parameter used within the adaptation algorithm. Convergence appears to be linear even through two adaptive steps. To further show the effect of applying adaptive algorithms to SOMA, the approximation is advanced further beyond 120 basis functions. Figure 5.4 presents

![Figure 5.3: Logarithmic plot of error progression in the adapted convection-diffusion equation through 120 basis functions adapting every 40 using the Basis Save method. The equation residual is shown as green circles and the approximation error as blue triangles. Linear and quadratic convergence labeled as such.](image-url)
the same information as Figure 5.3, however SOMA is allowed to run to 240 basis functions. The adaptation algorithm is implemented every 40 basis functions, where

![Graph](image)

Figure 5.4: Logarithmic plot of error progression in the adapted convection-diffusion equation through 240 basis functions, adapting every 40 using the Basis Save method. The equation residual is shown as green circles and the approximation error as blue triangles. Linear and quadratic convergence rates are labeled as such.

there are small dips in the scalar error value. At the beginning of this particular run, the approximation begins with 50 nodes evenly distributed. Between 50 and 100 nodes are added with each adaptation for a total of 652 nodes which are intelligently spaced to capture areas of high residual. The continuing convergence, albeit slow convergence, of the convection-diffusion equation is evidence that the adaptive algorithm is working as it should, without spoiling the convergence. Comparing this plot of error convergence to the unadapted mesh’s convergence from Figure 3.9 shows how adaptive methods can improve a stubborn convergence rate.
5.4 Adapting Poisson’s Equation

Poisson’s equation is another one-dimensional linear differential equation which can be
used to demonstrate the Basis Save method. Like the convection-diffusion equation,
for this example the derivatives are chosen to be calculated analytically. SOMA is
allowed to run for 20 basis functions and then the adaptive algorithm is applied.
Nodes are added near areas where the equation residual is greater than $10^{-5}$. The
final form of the approximation is shown in Figure 5.5.

![Figure 5.5: Plot of Poisson’s equation adapting every 20 basis functions for 100 basis
functions using the Basis Save method. The exact solution is shown as a red line
and SOMA’s approximation is shown as blue circles. The nodal distribution is shown
below the approximation as black circles. The nodes tend to congregate near the
boundaries where the equation residual is highest.]

The approximation is adapted every 20 basis functions for a total of 100 basis
functions. At this point the approximation was stopped because the mesh was be-
coming saturated and the results of the adaptation would become difficult to discern.
Notice how the nodes once again seem to congregate near the boundaries of the ap-
proximation, shown by the black circles below the approximation in Figure 5.5. The approximation is shown in blue, and the exact solution is shown as a red line. The distribution of nodes demonstrates how boundaries typically have a high equation residual compared to the rest of the domain. The approximation error and equation residual over the domain are shown in Figure 5.6.

Figure 5.6: Plot of the error over the domain of the adapted Poisson’s equation with the equation residual shown as green circles and the approximation error shown as blue triangles. The y-axis is multiplied by $10^{-5}$, which suggests that the error is very small; however, the typical boundary trend is still obvious.

The approximation error is much smaller than the equation residual throughout the domain. Like the convection-diffusion example, this demonstrates the difficulty SOMA has discerning the derivatives of the approximation verses the value of the approximation. To demonstrate how the adaptive algorithm improves SOMA, a semi-logarithmic plot of the scalar error is shown in Figure 5.7. The scalar error is calculated using Equation 3.17 for both the equation residual and the approximation error.
Figure 5.7: Logarithmic plot of the scalar error progression over the domain of the adapted Poisson’s equation. The approximation is adapted every 20 basis functions. The equation residual is shown as green circles, and the approximation error shown with blue triangles. Linear and quadratic convergence rates are labeled.

Once again, at least linear convergence is demonstrated throughout the adaptation. The adaptive algorithm applied every 20 basis functions helps to further push down the scalar error without hindering convergence. These two examples demonstrate how adaptive techniques in one-dimension work with SOMA. The following section demonstrates how these adaptive techniques can work in two-dimensions as well.

5.5 Adapting a Sine Function in Two-Dimensions

SOMA can be used to approximate any two-dimensional function, specifically the following sine function used in this thesis:

$$F = \sin(\pi x) \sin\left(\frac{\pi}{2} y\right).$$  

(5.7)
The domain used is $[-1, 1]$ for both $x$ and $y$. The initial approximation is a flat plane at zero, to which two-dimensional basis functions of the following form are added to the approximation:

$$
\phi = e^{w_N \left((x-x_{c,N})^2+(y-y_{c,N})^2\right)},
$$

(5.8)

This basis function takes the same form as the one presented in 5.4, except it now includes $y_{c,N}$ which is the $y$-coordinate of the basis function center. There is no need to calculate derivatives of this basis function because it is only used to approximate a linear, two-dimensional function. The approximation and its adaptations are compiled almost exactly like the convection-diffusion equation presented in Section 5.3, except without derivatives. Parameters are saved for each basis function and are then compiled into a large linear equation to which new nodes can easily be added.

Data representation changes for two-dimensional problems because their plots are three-dimensional representations. Displaying the location of the nodes as a flat plane with scattered data on top of a general figure showing the shape of the approximation seems to be a clear, well-used approach [1,24,25,38,39]. Figure 5.8 demonstrates the adaptive algorithm in two dimensions after 100 basis functions have been added. After every 20 basis functions, the adaptive algorithm is implemented. For function approximation, the algorithm seeks out the approximation error because there is no equation residual. The adaptive algorithm seeks out nodes with an error above $5 \times 10^{-2}$, and additional nodes are added in the cross pattern presented in Chapter 4.

The figure is set up so that the actual locations of nodes within the approximation are shown above at $F(x, y) = 5$. These blue dots represent only the locations of nodes in the $x$-$y$ plane. The mapping below shows the basic geometry of the approximation, but the mesh does not correspond to the actual mesh used in the approximation. Figure 5.8 shows how the algorithm will increase the fineness of the grid in areas of
Figure 5.8: A two-dimensional sine function after 100 basis functions with adaptations every 20 iterations using the Basis Save method. The locations of the actual nodes are shown as the blue plane of nodes above the colored surface which represents the sine curve.

the approximation where the residual is high and leave the areas of low residual alone.

The error progression for this particular test case deserves discussion. The normalized scalar value of the approximation error is calculated similarly to previous methods. The approximation error at each node is calculated by:

\[ r_n = f_N(x, y) - F(x, y), \]  

(5.9)

where \( r_n \) is the approximation error, \( f(x, y) \) is the approximation value at any particular node \((x, y)\), and \( F(x, y) \) is the exact value of the sine function at that same node. The normalized scalar error is calculated with the same equation used for the equation residual, Equation 3.17. Figure 5.9 shows the error progression through 100 basis functions with adaptation steps every 20 basis functions. The convergence rate for the sine function shows at least quadratic convergence after adaptation, which
Figure 5.9: Logarithmic plot of error convergence of the two-dimensional sine function through 100 basis functions with adaptations every 20 iterations. The approximation error is shown as green circles. Convergence rates are labeled as such.

demonstrates that even in two-dimensions, SOMA can be successfully adapted.

There are some drawbacks with the Basis Save method that can become apparent when originally sparse meshes are adapted and contain regions that become much, much finer. These problems could contribute to the error step-up in two-dimensional problems, and is explained in the following section.

5.6 Problems with the Basis Save Method

While exploring applications of neural network theory to this adaptation method, a pitfall of the Basis Save method was discovered [26]. More information about applying neural networks and training and test nodes to SOMA can be found in Appendix C.

The pitfall of this method can be mentally visualized with squeezing play-dough through a clenched fist, except the play-dough is the error within the domain and the fist represents the nodes within the original approximation. This intermediate error
shows up when trying to use the Basis Save method to apply a very fine mesh to an approximation that was developed using a much coarser mesh. Collocation methods such as SOMA seek to simply minimize error at each node locally, and do not work to decrease the overall residual if the equation of the approximation is applied globally to the entire domain. Regions of larger error can develop in the analytical approximation between the nodes that the approximation was constructed for [40].

The problem is best demonstrated using a simple one-dimensional sine function approximation. The equation approximated is

\[ F = \sin(2\pi x) \] (5.10)

SOMA is allowed to run without adaptation for 50 basis functions on a relatively sparse mesh of 21 nodes. The linear equation compiled using the Basis Saving method is then applied to a much finer grid containing 201 nodes. The results of this comparison are shown in Figure 5.10. Even though the actual approximations seem to match reasonably well, the approximation error demonstrates how SOMA does not reduce the residual evenly throughout the domain, but rather minimizes error only where there are nodes. The portion of the plot circled in black demonstrates the intermediate error quite nicely. Using a sparse grid to generate the basis functions for the approximation will do well for that sparse grid. When a finer grid is applied to those saved basis functions, regions of high error show up in-between the original, sparser grid.

Although the Basis Save method is very useful and applicable in many different scenarios, in situations such as that shown in Figure 5.10, interpolation can be used to solve the problems that arise when nodes are added. Chapter 6 demonstrates the usefulness of having an interpolation scheme determine the values of new nodes instead of calculating their values using a reconstructed approximation.
Figure 5.10: Plot of approximation error of a one-dimensional sine function calculated first on a sparse mesh then applied to a much finer one. Red nodes connected by the dashed line represent the 21 nodes for which the approximation was constructed. Blue nodes demonstrate the same approximation with 201 nodes. The circled area demonstrates how saving the basis functions of a sparse mesh can create regions of high error when a much finer mesh is applied.
Chapter 6

Interpolation Strategy

The second method of adding nodes to the approximation presented in this thesis is the interpolation method. Interpolation seeks to eliminate the problems associated with the Basis Save method. Like the Basis Save method, new nodes are added around existing ones which exceed some error tolerance discussed above. Determining the $x$ or $y$ location that these nodes should inhabit is determined the same way as presented in Chapter 4. The value of the approximation at the node as well as the values of their derivatives and boundary drivers make up much of the challenge of the interpolation method. Because the nodes adjacent to each new node are fully known, the new node can be placed along a line connecting the two adjacent original nodes. Various interpolation methods such as linear or cubic methods have been tested, and cubic methods have been chosen for this work.

Using interpolation schemes is effective for finding a new value of the approximation, but for many equations it is necessary to also find accurate values of the first and second derivatives of the approximations. From experimentation, it was shown that a simple linear interpolation of the derivatives does not suffice. The analytical and numerical derivatives demonstrated in Section 3.6 are much more accurate methods of calculating the derivatives compared to the interpolation strategy used on the value of the approximation. To fully differentiate this method from the basis save method, no basis functions were saved for the explanation of the method. All required derivatives were calculated using the numerical derivatives instead of analytical ones.
For reference, Section 3.6 contains a complete explanation of the different methods for calculating derivatives.

The following sections will first describe how the interpolation strategy is implemented followed by a discussion of how the derivatives and boundary drivers are handled. Next, some results are presented for this method. Finally, an alternative way of implementing this method is discussed, which is a combination of the Basis Save method and interpolation.

### 6.1 Implementing an Interpolation Strategy

SOMA is run as usual for a set number of basis functions, and the interpolation strategy begins when the adaptive algorithm is implemented. Any nodes with an equation residual above the user-defined tolerance are pegged for adaptation. New nodes are added around those high-residual nodes as outlined in Chapter 4, creating the new vector, \( x_a \). Unlike the Basis Save method, the vector \( x_a \) is not applied to the analytical approximation to determine its approximation values. In fact, because basis functions parameters are not saved, it is not possible to construct the analytical approximation for adaptation. Instead, the current approximation is used to interpolate the new nodes into position using a cubic interpolation scheme. These new nodes are then incorporated into the approximation as identical to the original nodes, and the approximation is continued treating the new nodes the same as the original ones.

Figure 6.1 demonstrates the addition of new nodes whose locations have been interpolated for the one-dimensional linear convection-diffusion equation. The approximation was initialized with 30 nodes and 11 new nodes were added around areas of high residual. The exact solution is shown as the red line and the blue circles represent the nodes within the approximation. The new nodes added to the approxi-
Figure 6.1: The convection-diffusion equation approximation with the added nodes using interpolation. The exact solution is shown as the red line, and the blue circles represent SOMA’s approximation. New nodes are filled in with magenta, showing areas where the error is high.

Approximation using the interpolation strategy are marked with the magenta triangles. This demonstrates how the interpolation strategy places these nodes in the approximation. The closeness of the approximation with the exact solution suggests that the approximation error is low. The equation residual, however, does not necessarily correspond to this seemingly good approximation error. The derivatives must be analyzed in order to determine the equation residual.

Determining the derivative values of these new nodes is a challenge that shows up repeatedly when adapting SOMA. The following section will demonstrate how the derivatives as well as the boundary drivers are handled in the adaptive algorithm.
6.2 Handling Derivatives and Boundary Drivers

The one-dimensional convection-diffusion example presented in the previous section uses the numerical derivatives of the approximation instead of the analytical derivative from the saved basis functions. Because the added nodes have interpolated approximation values, their derivatives can develop high values of error. This is especially prevalent in the second derivative where error mostly is propagated. The equations for the derivatives of the exact solution of the convection-diffusion equation can be found in Equations 3.15 and 3.16. Figure 6.2 plots the first and second derivatives of the adapted convection-diffusion equation from Figure 6.1 using numerical differentiation of the approximation.

Figure 6.2: The first and second derivatives of the approximation of the convection-diffusion equation. The exact derivatives are shown as red lines and the numerical derivatives are shown as blue circles.

It is obvious for the second derivative near $x = 1$ that the approximation does not match as well to the exact solution.

The boundary driver equations are not interpolated, instead they are just re-
initialized within the adaptation algorithm with the new nodes. This is the same way that the boundary drivers are handled in the Basis Save method. The interpolation method can be applied to any equation desired, although most of the testing was run using Poisson’s equation.

### 6.3 Interpolation Results

The following results stem from using strictly interpolation for the approximation values and their numerical derivatives. This means that no basis function parameters are saved anywhere in the algorithm. The one-dimensional Poisson’s equation is used as an example of the interpolation method. SOMA is started and allowed to run for 20 basis functions before the first adaptation step. Once again, using this method requires no saved basis functions and relies on interpolated approximation values and numerical derivatives. SOMA is continued and adapted every 20 basis functions for a total of 100 basis functions. The error tolerance begins at $1 \times 10^{-4}$ and is decreased after 60 basis functions to $5 \times 10^{-5}$ because of the decreasing error throughout the approximation. The approximation is started with 60 nodes. The final adapted grid contains 161 nodes specifically placed at areas of high equation residual.

Figure 6.3 shows the approximation after those 100 basis functions and the interpolation-based adaptation scheme. The distribution of nodes in the domain is shown by the black circles below the approximation and exact solution. Nodes congregate near the boundaries, and there are a few extras in the center of the approximation.

The error over the domain is shown in Figure 6.4. There are some areas of the approximation error which have relatively high error values, however their actual values are quite small. The scale of the $y$-axis is multiplied by $10^{-5}$, which means that these interpolated nodal values are only slightly off from the approximation.
Figure 6.3: Plot of the solution of Poisson’s equation after 100 basis functions using adaptive interpolation algorithm every 20 basis functions. The red line represents the exact solution and the blue circles represent SOMA’s approximation. Note the distribution of nodes shown by the black circles below the approximation.

The convergence history of Poisson’s equation is shown in Figure 6.5. The normalized equation residual and approximation error quickly reduce to a very small value, and the equation residual seems to demonstrate better than linear convergence. Convergence is not interrupted by interpolation, although the value of the approximation error does increase after 40 basis functions and stays at the higher value for the rest of the approximation. The increase in approximation error is due to the few nodes which have a high approximation error seen in Figure 6.4. The value of error is still very small, so this jump is not of great concern, but rather just demonstrates the difference in application between the Basis Save method and interpolation.

The flexibility of the adaptive algorithm makes it possible to investigate a combined approach to adaptation. This combined approach which merges both the Basis
Figure 6.4: Plot of the equation residual and approximation error over the domain of Poisson’s equation after interpolation-based adaptation every 20 basis functions for a total of 100 basis functions. The equation residual is shown as green circles and the approximation error is shown as blue triangles.

Save method and interpolation, is discussed in the following section.

6.4 A Combined Interpolation Strategy

Even if interpolated approximation values are desired, the analytical derivatives of the approximation can still be used. This requires that the basis function parameters be saved for each iteration so that the analytical derivatives can be reconstructed and used to determine the derivative values of the new nodes in the domain. This combination of the Basis Save method and interpolation works at least as well as SOMA without adaptation, of course with the added benefit of higher resolution in areas where it may be important. This combined method will be called the Semi-Interpolation method.

This Semi-Interpolation method was tested first by using the convection-diffusion
Figure 6.5: Logarithmic plot of the normalized equation residual and approximation error over the domain of Poisson’s equation after interpolation-based adaptation every 20 basis functions for a total of 100 basis functions. The equation residual is shown in green and the approximation error is shown as the blue triangles. Linear and quadratic convergence are labeled.

The results are at least as good or better than the convection-diffusion equation without adaptation. Results for that test as well as other equations tested are outlined in the following sections.

### 6.5 Semi-Interpolation Convection-Diffusion Results

The first equation tested was the one-dimensional linear convection-diffusion equation. This has been a baseline case throughout work on this thesis due to its ease of implementation and for comparison purposes between methods. Poisson’s equation is also tested using the Semi-Interpolation method.

To begin the convection-diffusion equation, SOMA is run for 20 basis functions before the first adaptation is applied. It is then adapted again at 40, 80, and 120
basis functions. The final approximation is stopped at 200 basis functions, at which time 12 nodes were added to the original 50 for 62 nodes in the approximation. The following figures show the approximation, its error, and its normalized error progression. Figure 6.6 shows the approximation in its final form after 200 basis functions have been added.

Figure 6.6: Plot of the convection-diffusion approximation with added nodes using the Semi-Interpolation method of adaptation every 40 or so basis functions for a total of 200 basis functions. The red line demonstrates the exact solution and blue circles represent SOMA’s approximation. Black circles show the nodal distribution of the adapted approximation.

The error for this particular approximation is shown in Figure 6.7. The majority of the error of the equation residual is congregated around the boundaries, especially near \( x = 1 \), where the slope of the approximation increases. Like other plots, the equation residual is shown in green and the approximation error in blue.

Figure 6.8 demonstrates the normalized error convergence for both the equation residual and the approximation error. From this plot, it is clear that the Semi-
Interpolation method maintains slow convergence through the addition of nodes.

Another example of how the Semi-Interpolation method is implemented is demonstrated using Poisson’s equation.

6.6 Semi-Interpolation Poisson’s Equation Results

The one-dimensional linear Poisson’s equation is adapted using the same Semi-Interpolation strategy as the convection-diffusion equation. The approximation is run to a total of 200 basis functions, adapting every 20 until 120 basis functions. It is allowed to run without adaptation until 180 and then finally is stopped at 200. The error tolerance for adaptation changes from $10^{-4}$ to $10^{-5}$ as the error decreases.

Figure 6.9 shows the approximation after 200 basis functions. The approximation shows the results of adaptation, there are nodes congregated near the boundaries and
in some center locations where the error was high.

The error over the domain is shown in Figure 6.10. The equation residual is higher than the approximation error, however both are very small. In this plot, there are a few nodes that have a higher approximation error near $x = 0.9$. These small jumps in the approximation error are only observed when interpolation methods are used, but do not seem to present a problem for the algorithm as a whole.

Convergence of the normalized equation residual and approximation error is shown in Figure 6.11. The normalized approximation error, shown as blue triangles, reduces to approximately $10^{-7}$ after about 20 basis functions, and stays very low. The normalized equation residual continues converging after the first adaptation step until the algorithm is stopped at 200 basis functions. The normalized error convergence plot for Poisson’s equation once again shows a continual convergence until the algorithm
Figure 6.9: Plot of Poisson’s equation After 200 basis functions adding nodes about every 20 basis functions using the Semi-Interpolation method. The red line represents the exact solution, and the blue circles are SOMA’s approximation. Black circles represent the nodal distribution after adaptation.

Figure 6.10: Plot of the equation residual and approximation error of Poisson’s equation after 200 basis functions using the Semi-Interpolation method.
Figure 6.11: Logarithmic plot of the normalized equation residual and approximation error of Poisson’s equation. The equation residual is shown as green circles and the approximation error as blue triangles. Linear and quadratic convergence are labeled.

is stopped by the user or within the algorithm itself. These convergence plots suggest that as long as the method used to calculate the derivatives and the values of the approximation are good enough, various methods can be used with similar results.

Both the Basis Save method and variations of interpolation methods can be effectively used in SOMA to create an adaptive approximation. Depending on the amount of time and memory capacity the user desires to use for the approximation, any combination of methods can be used. Chapter 7 will conclude this thesis as well as discuss suggestions for future work pertaining to adapting SOMA as well as other features of SOMA which should be explored.
Chapter 7

Conclusions and Future Work

Residual-based adaptive mesh refinement techniques have been successfully applied to the meshless solver SOMA. The first challenge of adapting a previously fixed-grid approximation is determining where nodes should go in the domain. The adaptive algorithms of this thesis begin by seeking out areas of high residual within the domain. The residual was chosen as the primary metric of adaptation since it is already calculated at every iteration by SOMA no matter which equation SOMA is trying to solve. However, the flexibility of the adaptive techniques developed in this thesis does not preclude the use of gradient-based methods.

Nodal refinement was chosen due to its ease of implementation in collocation-type methods as well as the substantial documentation in previous literature of its efficacy. A technique similar to edge-bisection from finite elements is used to determine the new vector full of original and adapted nodes. Nodes are added near existing nodes of high residual. In one-dimensional problems, two nodes are added on either side of the existing nodes. In two-dimensional problems, four nodes are added in a cross pattern. Once this first step of determining where the nodal distribution should be refined and the locations of these new nodes is complete, the second challenge of adaptation begins.

The second challenge is determining the new nodes’ approximation values as well as their derivatives. Two different methods of adaptation are presented. The first uses the saved parameters of each basis function in order to reconstruct the analytical
approximation and its required derivatives. This Basis Save method requires saving a matrix of parameters to build a linear approximation of the equation of interest. The second method relies solely on interpolation of the locations of the new nodes and the numerical derivatives of the approximation, so no basis function parameters need to be saved. Because the method of adding nodes to the domain does not change, these methods can be combined in order to create various methods each of which has its benefits and drawbacks.

These methods are applied to various one- and two-dimensional functions and differential equations for proof of concept. Although no three-dimensional tests have been run using adaptive schemes, the methods are easily modified to account for additional dimensions. Convergence, albeit sometimes slow convergence, is shown for all example equations with the adaptive algorithm. These adaptive methods are therefore able to refine areas of the approximation which need refinement without resorting to a fully dense grid.

7.1 Future Work

SOMA exists as a jumping off point from which many new applications and new methods can be tested. Already, there is a large body of work that has been performed applying SOMA as a CFD solver and applying it to equations such as the Fokker-Planck equation [4, 36]. All of this research builds momentum and credibility for the method and every successful implementation of SOMA opens the doors for even more applications and tests that can be done.

The future work discussed in the following chapters will focus on new adaptive methods, as well as suggest further applications of the methods presented in this thesis.
7.1.1 More Adaptation Work

Within the adapting meshing discipline, there are countless methods designed to be faster, more robust, and more accurate. Further work should focus on testing different adaptive techniques from the ones in this thesis. These different techniques range from nodal movement techniques to nodal deletion and could possibly solve some of the data storage and approximation accuracy problems that the presented method has shown. For example, Šarler and Vertnik [41] present a method in their 2006 paper which overcomes problems with large scale “bottlenecks” associated with other methods. Their globally unsupported collocation method works locally over a set of smaller domains. This sub-division of the domain was also suggested by Wilkinson as a useful addition to SOMA [5].

Although different adaptive schemes are certainly possible, exciting future adaptive work should also focus on the problem of expanding the adaptive techniques presented in this thesis to the full range of CFD capabilities that SOMA has. At the time of writing, adaptive techniques have not been applied to any three-dimensional test or validation cases because those tests are still in their debugging stages. Once these simulations are fully capable using a fixed grid, adaptation of these more advanced simulations is the next logical step towards creating a fully robust algorithm. The adaptation of three-dimensional test cases is important especially for capturing flow structures such as shocks without requiring uniformly dense grids [5].

Future explorations of the method of calculating the coefficient seem necessary, especially considering the promising results shown in Appendix A. Some work has been done to determine how to best calculate the coefficient of each basis function for other RBF methods [42], however only preliminary work has been conducted in SOMA with regards to adaptation.
The methods of this thesis seek to add new nodes into the approximation without interrupted error convergence. However, adding nodes into the approximation with the intent of refining the mesh is not the only reason that nodes may be added to the domain.

### 7.1.2 Experimental Data Integration

The work done in this thesis has shown that technically, nodes can be added anywhere in the domain or the range of interest. Therefore, it is possible to choose any value for new approximation nodes although it may not be advantageous to place nodes haphazardly around the approximation. There is, however, another source of approximation data that could be used effectively and presents an important method of validating results from SOMA. By incorporating experimental data into the approximation, SOMA can be improved and validated as an effective CFD tool.

Incorporating data into an approximation using SOMA would require that either the initial domain, or the domain after a certain number of basis functions, contain nodes whose values are fixed at these experimental values. It is then assumed that these experimentally determined approximation values will help SOMA converge on a solution, prove SOMA’s accuracy as a CFD solver, or both. This incorporation of experimental data sounds very similar to the methods presented in this thesis. Instead of placing nodes near areas of high residual and giving them calculated approximation values, a new manner of placing nodes would have to be developed. The nodal location would be determined from experimental setup, and the value of interest of each node would be determined from the experimental data associated with that location.

Experimental data integration would really only be useful for a version of SOMA which calculates flow parameters over airfoils or on objects which have accepted
and validated experimental results. This thesis demonstrates clearly that nodes can be successfully added to SOMA's approximation within the algorithm, so in theory adding experimental data would be a next logical step.
Bibliography


Appendices
Appendix A

Optimized Coefficients in SOMA

Typically, the coefficient of each basis function, $c_k$, is a calculated parameter in SOMA. Almost serendipitously, an optimized coefficient routine was implemented and discovered to be a great alternative to the calculated coefficient for some problems. The rest of this appendix will describe how the coefficient is split into two parameters for optimization and will present some results of using calculated coefficients versus optimized ones.

A.1 Splitting the Coefficient

The coefficient as an optimization parameter can be troublesome because in practice it can vary between $-10^5$ and $10^5$. Sometimes its value can even fall beyond that range. This means that the optimization routine within SOMA has a large range in which search for the optimized coefficient. One approach to working around this is to represent the coefficient slightly differently within the optimization routine. The coefficient can be split up into two parameters and represented with the following equation:

\[ c_k = p_{1,k} \cdot 10^{p_{2,k}}, \quad (A.1) \]

where parameters $p_{1,k}$ and $p_{2,k}$ are optimized by the genetic algorithm. Using two different optimization parameters works nicely in SOMA to hone in on a truly optimal
coefficient for each basis function. The parameter $p_{2,k}$ works to find a coefficient that is of the correct order of magnitude for the problem. The parameter $p_{1,k}$ can be thought of as a “finesse” parameter because it hones in the most optimal value for the coefficient.

### A.2 Results of Optimized Coefficients

While it would seem as though the calculated coefficient may work better than one determined from an optimization routine, the results speak for themselves. Once again, the one-dimensional convection-diffusion equation is used as an example. The first set of plots shown below represent SOMA with the optimized coefficients as outlined above with Equation A.1 used as the formula for the coefficient with two optimization parameters. The following plots show SOMA as it originally is configured with calculated coefficients for comparison. For both experiments, 20 nodes are placed evenly throughout the domain and the error tolerance which stops SOMA is set to $10^{-4}$. Figure A.1 shows the value of error over the domain with optimized coefficients after 573 basis functions. The green circles represent the equation residual and the blue triangles represent the approximation error. The y-axis is multiplied by $10^{-5}$ in this figure which represents extremely low error values especially compared to the alternate calculated coefficient method. This plot represents a snapshot of the error at the point when SOMA ended the algorithm because the final error tolerance was reached. The convergence plot for the convection-diffusion equation with optimized coefficients is shown in Figure A.2 as a semi-logarithmic plot of the normalized error for each basis function. The approximation with optimized coefficients converges in fewer basis functions as well as in less time than using the calculated coefficient method. This particular run took 70 seconds as measured from start to final conver-
Figure A.1: Plot showing the equation residual and approximation error over the domain of the convection-diffusion equation after 573 basis functions using optimized coefficients. Note the y-axis scale is multiplied by $10^{-5}$, which represents very small values of error for each node.

Gence. Also, it is clear from Figure A.2 that the progress does not show much sign of slowing down.

Alternatively, the behavior of SOMA when coefficients are calculated using Equation 3.7 seems to be sub-optimal. Figure A.3 demonstrates the error over the domain of the exact same set-up of SOMA except that the coefficients are now calculated instead of included in the optimization routine. The routine is stopped at 573 basis functions for comparison. After the full run, the normalized equation residual has a value of 0.1096, which is much higher than $1 \times 10^{-4}$ (the value of tolerance that the optimized coefficient method obtained). The convergence plot while using calculated coefficients is shown in Figure A.4. Neither the equation residual or approximation error converge to the tolerance which is again labeled as a dashed black line. It also appears that they probably will not converge in any sort of timely manner. This
Figure A.2: Logarithmic plot showing the progress of the equation residual and approximation error over the domain of the convection-diffusion equation after 573 basis functions while optimizing the coefficients. The error tolerance, shown as a red dashed line at $y = 10^{-4}$, is reached at 573 basis functions which ends SOMA.

calculated coefficient run took 207 seconds to run to 573 basis functions, so it both took longer and converged less.

Unfortunately, this new idea of optimizing coefficients was only put into use very recently (about a week before this writing), so not much work has been done in this particular area besides convergence plots of fixed-grid approximations. Future work on SOMA will certainly include work on the coefficient to create the best SOMA possible.
Figure A.3: Plot showing the equation residual and approximation over the domain of the convection-diffusion after 573 basis functions with calculated coefficients. Note how much larger the y-axis scaling is which demonstrates how the equation residual stays much higher when the coefficient is calculated.

Figure A.4: Logarithmic plot showing the progress of the equation residual and approximation error over the domain of the convection-diffusion equation after 573 basis functions with calculated coefficients. The error tolerance, shown as a red dashed line at $y = 10^{-4}$, never reached and the obvious trend of the equation residual suggests that the tolerance may never be met.
Appendix B

Gradient and Sparseness as Adaptation Metrics

Throughout this thesis, the equation residual has been the primary metric driving the adaptation. The residual is not the only choice of metric possible, although it is a common choice [3, 26, 28]. Another common choice, especially for adaptive finite element methods, is the gradient of the approximation [7, 39]. This appendix will also explore the use of sparseness, or the distance between adjacent nodes, as a metric of adaptation.

B.1 Gradient as an Adaptation Metric

Regions of quick changes in the domain of an approximation often lend themselves to higher error values and therefore warrant adaptation. In SOMA, the gradient of the approximation can be calculated using a couple of different techniques, outlined in Section 3.6. Using these techniques, a value of the approximation gradient can be determined for each node. The adaptive algorithm can then be applied using the gradient as its metric instead of the residual. Of course, this will require a change in the tolerance. Often this tolerance is set based on a priori knowledge of the values of gradient and captures the largest values of gradient without over-saturating the approximation.

The gradient is calculated before adaptation using the numerical derivatives of the approximation as discussed in Section 3.6. Numerical derivatives are used so that the...
basis functions only have to be compiled once after adaptation to determine the values of the derivatives of the new nodes. If the analytical derivatives were desired for determining where the approximation should be adapted, the basis functions would have to be compiled before and after the adaptation. Two compilations would be necessary to determine where the gradient is high as well as to determine the derivatives of the new nodes once they have been added. To save computational time, numerical derivatives are used instead.

An evenly distributed mesh of 60 nodes is run through 20 basis functions of SOMA using Poisson’s equation. The numerical derivative of the approximation is calculated, and the areas of high derivative value are subjected to adaptation. Then SOMA is continued to 40 basis functions. Figure B.1 shows the approximation of Poisson’s equation after those 40 basis functions. Note how the adaptive algorithm adds nodes in regions of high gradient, near the boundaries but not near the center.

The error over the domain after 40 basis functions is shown in Figure B.2. The error seems to be similar to the error using residual-type methods. This suggests that gradient methods are viable alternatives to residual methods when applied to SOMA. They don’t seem to work significantly better or any worse than residual methods.

Finally, the error convergence is shown in Figure B.3. The equation residual and approximation error both reach very low values relatively quickly. The adaptation scheme does not seem to interrupt convergence, and the final errors are of similar orders of magnitude to residual methods.

### B.2 Sparseness as an Adaptation Metric

Occasionally, it may be useful to refine regions solely based on the distribution of nodes in that area. One of the methods tested with SOMA was adapting the ap-
Figure B.1: Plot of the distribution of nodes over the approximation of Poisson’s equation. Nodes are congregated where the gradient is highest. The red line represents the exact solution, and the blue circles represent SOMA’s approximation. The black circles below represent the distribution of nodes within the domain.

Figure B.2: Plot of the error distributed over the domain of Poisson’s equation after adaptation using gradient adaptation. Green circles represent the equation residual and the blue triangles represent the approximation error.
Figure B.3: Logarithmic plot of the equation residual in green and approximation error in blue after each basis function. The equation residual is shown as green circles and the approximation error as blue triangles. Linear and quadratic convergence are labeled as such. After 40 basis functions, it seems like SOMA will continue to push the error down even after adaptation.

approximation using sparseness as the primary metric for adaptation. This method is particularly useful if the user suspects that a sparse region may be hindering the solution, or just for extra fidelity at a certain area.

To demonstrate how SOMA can be adapted based on nodal distribution, the one-dimensional, linear Poisson’s equation is used as an example. The mesh is initialized with an uneven mesh. From $0 < x < 0.5$, the distribution is relatively dense, and from $0.5 \leq x < 1$, the distribution is significantly sparser. The original nodal distribution is shown in Figure B.4. Notice how the nodal distribution is initially uneven. This uneven distribution is intentional to demonstrate how the algorithm can add nodes only where the approximation is deemed “sparse.” In this case, the distance between nodes that is considered too sparse corresponds to adding nodes to the right-hand
side of the approximation shown in Figure B.4.

Figure B.4: Plot of the initial distribution of nodes used for Poisson’s equation. Note how the nodes are relatively dense on the left side of the approximation, and relatively dense on the right.

The approximation is advanced through 20 basis functions, and then the adaptation algorithm is applied. The Basis Save method from Chapter 5 is used to determine the values of the new approximation nodes and their derivatives. SOMA continues until a total of 40 basis functions have been added. Figure B.5 demonstrates the approximation of Poisson’s equation after those 40 basis functions. As usual, the distribution of nodes is shown as black circles below the approximation, but this plot includes the locations of the new additional nodes with filled-in magenta triangles.

Residual plots of the Poisson’s equation using this sparseness metric are very similar to the ones used for residual-based methods. Figure B.6 shows the equation residual and approximation error over the domain of the approximation. This error plot over the domain suggests that using sparseness as an adaptation metric pro-
Figure B.5: Plot of the adapted distribution of nodes used for Poisson’s equation. The locations of nodes is relatively even, as shown by the black circles below the approximation. The magenta-filled circles represent the additional nodes.

Figure B.6: The equation residual and approximation error over the domain of Poisson’s equation after sparse adaptation. The equation residual is shown as green circles and the approximation error as blue triangles. The order of magnitude of the error is very low, and similar to the other adaptation schemes.
duces similar results as using equation residual or gradient. The convergence plot for this sparseness method shows similar results to other methods. Figure B.7 shows the error progression over the 40 basis functions. Recall that the adaptive algorithm was applied at 20 basis functions. Although sparseness is clearly a usable metric for adaptation, using it does not always provide the same amount of inherent approximation data as gradient or error. Using sparseness to try to capture a shock wave or boundary layer may not work so well because it does not take into account regions of high variation error within the approximation. For this reason, sparseness is not included in the main analysis of this thesis, but is rather presented as another proof of concept for adapting SOMA.

Figure B.7: Logarithmic plot of the equation residual and approximation error after each basis function. The equation residual is shown as green circles and the approximation error as blue triangles. Linear and quadratic convergence are labeled as such. After 40 basis functions, it seems like SOMA will continue to push the error down even after adaptation.
Appendix C

Training and Test Nodes

While exploring different applications and ways of using the Basis Save method, a slightly different method of adding nodes to the approximation was suggested and tested. The method is called the Training and Test Node method, because it utilizes a sparse grid to “train” SOMA to a correct solution, and a finer “test” grid to determine the progress of the solution. The two-dimensional function approximation problem presented in Section 5.5 is used as an example of the Training and Test Node method.

Similar training and test node methods have been widely researched and are shown to be great methods for solving various differential equations [26, 43–45]. Chen et al. present an integrated radial basis function network. nodes are added or removed on a relatively sparse grid by using the residual which is evaluated at a finer node set [25, 26]. Similar techniques are applied to SOMA, and the method used to apply these techniques is outlined below.

C.1 Function Approximation Example of Training and Test Nodes

First, SOMA is run with the Basis Save method (it saves the parameters generated for each basis function) out to a certain number of basis functions that the user defines. The grid initially used is the sparse, training grid. After the final basis function, the algorithm pauses with the saved linear equation for the approximation. New values
of $x$ and $y$ which comprise a dense test grid are then applied to the linear equation constructed from the sparse grid. Figure C.1 shows an example training and test grid. These are the two grid distributions used for the example in this Appendix. The original training grid contains 361 nodes, and the test grid contains approximately ten times the number of nodes with 3481. The function being approximated is the same sine function used in Section 5.5.

Figure C.1: Two plots showing the difference in distribution between training and test grids. The training grid is shown on the left and is the sparser grid. Training grids are typically much more dense, shown on the right. The colors of this plot simply represent the location of each node in the $z$-axis.

These two grids are connected by the analytical approximation of the function, however the difference between the training and test grids is how the adaptive algorithm is implemented. Instead of using the adaptive technique presented in Chapter 4 where nodes are added in a symmetric pattern around nodes of high error, these nodes are selected directly from the test grid to be incorporated into the matrix of nodes comprising the training grid. The Training and Test Node method seeks out
nodes in the test grid that exceed a similar error tolerance to other methods presented in this thesis. Once the location of these nodes are determined, nodes at these locations are added to the training grid and the approximation is continued.

In this example, the SOMA begins work on the sparse training grid of 361 nodes for 20 basis functions. After 20 basis functions, the dense test grid of 3481 nodes is applied to the approximation created from the training nodes. The adaptive algorithm seeks out areas of approximation error higher than $5 \times 10^{-2}$ in the dense test grid. These nodes with high error are then added to the training grid for further refinement. This process is repeated every 20 basis functions, and Figure C.2 shows the results after 100 basis functions.

![Figure C.2](image)

Figure C.2: Plot of the training grid after 100 basis functions, adapting every 20 basis functions. The blue nodes above show the distribution of nodes over the domain, and the three-dimensional plot below demonstrates the shape of the approximation with the initial nodal distribution.

The areas where the approximation error was high in the test grid are obvious by looking at the clusters of blue nodes above the three-dimensional training grid.
The training grid shown is the original nodal distribution before any adaptation took place. It is clear how sparse the grid was, especially when compared to the actual test grid. The actual test grid after 100 basis functions is shown in Figure C.3. It is important to note that the figure has been rotated in order to demonstrate how it is related to the training grid. The arrow in Figure C.3 points to an obvious dent in the grid where the error is high. The algorithm used to construct the training grid does not seek to minimize the error at nodes between those already in the training grid. So when the sparse grid’s equation is applied to a dense grid, areas of high residual show up as dents or bumps in the mesh. Occasionally, these dents or bumps can become very large compared to the grid and are further proof of the problems associated with the intermediate error effect discussed in Section 5.6.

An error log of the training grid is shown in Figure C.4. The normalized error
continuously decreases almost quadratically throughout the algorithm, with slight jumps where nodes are added. The normalized error associated with the training grid

Figure C.4: Logarithmic plot of the normalized approximation error progression of the training grid. Linear and quadratic convergence are labeled as such. Nodes are added every 20 basis functions and corresponds to slight jumps in the error.

is a good metric of how the grid is progressing, but perhaps the more important error metric is the normalized error of the test grid after each adaptation. The normalized error progression of the test grid is shown in Figure C.5.

It is important to remember that SOMA is not working on the test grid, but rather the training grid. So convergence on the test grid is a good way to determine how well the adaptation on the training grid is working. The continual decrease in error between adaptation steps is expected, and demonstrates the potential of using training and test grids. Occasionally some problems will arise while using this method to adapt SOMA and are outlined below.


Figure C.5: Logarithmic plot of the normalized approximation error progression of the test grid after every adaptation. Linear convergence is labeled.

C.2 Problems with the Training and Test Node Method

The heart of the Training and Test Node method is the Basis Save method because the fine grid is populated using the saved basis functions from the sparse grid. For this reason, it follows that the problems that arise when using the Basis Save method will also appear using the Training and Test Node method. Section 5.6 presents the intermediate error effect which describes how SOMA only seeks to minimize the error at nodes included in the approximation, but not globally throughout the domain. The same effect is sometimes seen when applying training and test nodes to SOMA. The algorithm is advanced using the sparse training grid, but when the analytical form of the approximation is applied to the test grid, regions of relatively high error pop up. Often these regions are one or two nodes within the test grid that can have a very high error compared to the rest of the test grid.

The following plots demonstrate this problem with the Training and Test Node
method. For this particular test, SOMA began with a sparse training grid of 361 evenly spaced nodes. The test grid used is much finer and contains 9801 nodes. The training grid was allowed to run for 60 basis functions before the test grid was applied. Figure C.6 shows the fine test grid after those 60 basis functions. The arrows point to regions of very high error where the intermediate error becomes apparent.

Figure C.6: Plot of the finer test grid approximation after 60 basis functions performed on the very sparse training grid. The arrows point to regions of very high error which demonstrate the intermediate error effect.

Figure C.7 shows the error plot of the same approximation from Figure C.6. The black arrows once again point to the corresponding high values of error that show up in the test grid.

The effect seems most apparent when the difference in the number of nodes between the training and test grid is high. Also, when more basis functions are used in the training grid before applying the test grid adaptation, the more regions of high error will show up in the fine test grid. From the results, it appears that as long as
Figure C.7: Plot of the finer test grid error after 60 basis functions performed on the very sparse training grid. The arrows point to regions of very high error which demonstrate the intermediate error effect.

If the test grid is applied often enough and the training grid adapted accordingly, these error problems can be avoided.