BUILDING INFRASTRUCTURE FOR MULTIPHYSICS SIMULATIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Michael James McCourt

January 2013

BUILDING INFRASTRUCTURE FOR MULTIPHYSICS SIMULATIONS

Michael James McCourt, Ph.D.

Cornell University 2013

As the use of computer simulation for scientific discovery increases there is a growing need for reliable multiphysics simulations. Although an exact definition for multiphysics problems is difficult to state, these problems include simulations where two or more component models are coupled to simulate events beyond either individual component. Aware of the growing prevalence of multiphysics simulations, this work identifies potential impediments to efficient and stable computation and proposes procedures to address the concerns.

We first introduce the implicitly coupled multiphysics framework, through which we limit the domain of problems we consider, and Jacobian-free Newton-Krylov methods, which will be the primary technique for solving these nonlinear systems. Kernel-based approximation theory is introduced, providing a method for coupling component models with different discretizations. These topics have both theoretical and computational implications which are later applied in the context of multiphysics simulations.

Our first contribution is the analysis of preconditioning nonlinear systems produced during multiphysics simulations. The motivating application is computational plasma physics, specifically, the edge region of a tokamak reactor performing magnetic confinement fusion. For sufficiently difficult parameter choices, existing preconditioners have proved ineffective or inefficient at producing useful Newton steps. We analyze the various components of this simulation and determine why preconditioners acting on the entire simulation do not perform well. Using these

insights we develop an operator-specific (or physics based) preconditioner which allows for better performance while improving parallel scalability.

Preconditioning is an important component of the Jacobian-free Newton-Krylov method because it allows a faster solution of the Newton search directions. Equally critical, although less susceptible to poor performance, are the Jacobian-vector products within the linear solvers which are approximated using Taylor series. We study this process in the multiphysics setting and propose improvements that allow for greater accuracy when two components on different scales are coupled together.

The final issue we consider in multiphysics systems also receives the most treatment. Simulating a multiphysics system requires coupling between the individual components, and in this thesis we discuss the use of kernel-based scattered data interpolation to perform the coupling. A new technique, based on an orthonormal expansion of the kernel, is developed which allows us to evaluate meshfree radial basis interpolants in arbitrary dimensions without the ill-conditioning often present for accurate kernel choices. These eigenfunctions are derived using Hilbert-Schmidt theory, and tested on interpolation problems in up to five dimensions.

After the eigenfunction method is validated, we show how it can be used to approximate derivatives of a function given only scattered data. Once this is proven, results are given for a multiphysics simulation using meshfree interpolation. These results are compared to the standard discretization scheme for interpolating between models, with higher order results possible using meshfree interpolation. Additionally, because the meshfree approach works with scattered data, it provides a more general method for coupling two models with mismatched grids. In addition to the viability of meshfree coupling for multiphysics, we also consider the computational cost. The preconditioning discussion from earlier is applied here to

choose a good preconditioner for the fully coupled system, which is important for reducing the cost of linear solves.

The remaining content focuses on other aspects of the eigenfunction expansion which are relevant to the wider computational community. Using the derivative approximation method created for coupling multiphysics components, we show how boundary value problems can be solved by collocation using the eigenfunction basis. We also explore the use of the eigenfunction technique in the Method of Particular Solutions, which demonstrates the benefit of solving elliptic problems with a joint collocation/Method of Fundamental Solutions approach. The final results deal with a statistical framework for determining appropriate parameterizations of kernels, which is necessary to realize the optimal interpolation accuracy discussed earlier. These methods have previously suffered from the ill-conditioning addressed by the eigenfunction approach, and, using the new stable basis, we reconsider their viability as predictive tools.

BIOGRAPHICAL SKETCH

Michael McCourt was born in Brunswick, OH in 1985, where he lived until he turned 18 and began studying applied mathematics as an undergraduate at the Illinois Institute of Technology. It is there that he began working on numerical analysis, both in the classroom and in a research setting. During his four years in Chicago, he had the privilege of working on projects ranging from radial basis function parameterization, random number generators on fractal sets, spectral collocation methods for reaction-diffusion systems and high performance computational plasma physics. After graduating in 2007, he went to Cornell to begin graduate study in applied mathematics. After receiving a M. S. degree in 2009, he began a lab grad research project at Argonne National Laboratory as part of his Ph. D. research.

This document is dedicated to the hustle.

ACKNOWLEDGEMENTS

I would like to express my appreciation to my advisor, Dr. Charles Van Loan, for his flexibility and encouragement in allowing me to pursue research outside of Cornell. I also need to thank him for reaching outside of his core research areas to help me conduct this research and write this thesis. I would also like to thank Dr. Lars Wahlbin, Dr. Derek Warner and Dr. Phaedon Stelios-Koutsourelakis for serving on my special committee. Also at Cornell, Selene Cammer and Dolores Pendell have helped handle all the logistical difficulties associated with my Argonne/Cornell relationship, and this work could not have happened without their help. Thank you also to the NSF graduate research fellowship program, for their support and for helping me attend Cornell University.

At Argonne National Labs, I would like to thank Dr. Hong Zhang and Dr. Lois Curfman McInnes for their tireless support, both mental and financial, in completing this thesis; your belief in me at times exceeds my own, and I am indebted to you for your guidance over the past 7 years. Satish Balay and Dr. Barry Smith also deserve credit for their support on numerous issues while I was working at Argonne. I also want to thank the Fusion LCRC crew for helping me run simulations on their high performance machine. And last, but certainly not least, I have to thank (soon to be Dr.) Sean Farley, my lab grad compatriot for always giving me something to think about.

Tom Rognlien is a collaborator who I thank for his patience while explaining plasma physics to me, and also for his support as a veteran of the research lifestyle trying to help me get by. I want to thank Scott Kruger, Ammar Hakim and John Cary for allowing me to participate in the FACETS program, and for including me in several valuable meetings with the top researchers in magnetic confinement fusion.

Dr. Greg Fasshauer, at different times a professor, a collaborator and a friend, has been fundamental to parts of this thesis; I give both my thanks and my best wishes going forward. While away from Cornell, I was still able to teach classes because Dr. Fasshauer and Dr. Fred Hickernell supported my position as an instructor at the Illinois Institute of Technology. I thank them for that, as well as allowing me to participate in useful seminars where I was able to flesh out some ideas and practice presenting.

Countless hours on the telephone with Dr. Graeme Fairweather have helped me make wise decisions about my research goals and future plans. I want to thank him for his unending kindness and support in this nascent stage of my career. I also want to thank Dr. Thomas Hangelbroek for inviting me to an AMS meeting at his university in Honolulu where I was able to develop key parts of this thesis. I would also like to thank David Keyes, with whom I had several wonderful conversations that reminded me how much I love this research and how much I wanted to continue. On that note, I should thank the entire computational science research community for being populated with the most unselfish, generous and helpful people anywhere.

Finally I need to thank my family, Laurie McCourt, Jason McCourt and Kevin McCourt for their support during this process. It has been a long and confusing road, but we are finally here, and I certainly would not have made it without your belief and that voice in my ear perpetually telling me "you'll figure it out". Now we are the Doctors McCourt.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

There exists significant mathematical theory and computational means to discretize and solve a set of partial differential equations (PDEs) based on a single model, or a single set of underlying physical phenomena. Currently, applications scientists are interested in performing simulations with multiple independent components coupled together. Techniques to run such multiphysics simulations have been developed within individual communities interested in the results, but numerous complicated issues remain for general multiphysics simulations [106, 28, 160, 175].

When using the term multiphysics in this work, we mean specifically the coupling of two simulations through an interface, although that interface may be the entire domain. Other works discussing multiphysics take a more targeted view (the component simulations may need contradictory views of the universe), or a more relaxed view (arbitrarily many simulations may be coupled in some way which varies with time) but we maintain a specific focus in this work to identify systems for which these new contributions are relevant.

Often times multiphysics problems are coupled together through an operator splitting allowing individual physics to be described separately. This is the natural extension of a bottom-up approach of assembling component models into a coupled simulation. Doing so introduces both algorithmic and analytic issues with several possible resolutions; many of those already present in existing applications is discussed to varying degrees in this thesis. Here we mostly consider problems which are intrinsically coupled, and whose components are asymptotic limits of parameters within the coupled simulation. Analysis on "real world" problems is

not always practical, both in production level simulations and in this thesis, so the use of model problems is needed.

This chapter introduces some existing multiphysics problems and related model problems as well as our preferred approach for solving these nonlinear problems, Jacobian-free Newton-Krylov. Kernel-based interpolation is also introduced, as this is the focus of several contributions in this thesis, most notably in the coupling of multiphysics simulations. A physics-based approach to preconditioning the Jacobian-free Newton-Krylov solver is developed in Chapter 2 for edge plasma simulation in a magnetic confinement fusion reactor. Though it is focused specifically on the plasma physics application, this preconditioning strategy is applicable elsewhere, including examples later in this thesis. Chapter 3 considers the error associated with finite difference matrix-vector products for linear solves of multiphysics problems.

The following two chapters transition away from multiphysics centric content towards kernel-based approximation theory, specifically, using Gaussian kernels. Chapter 4 develops new techniques for stably approximating the solution to Gaussian interpolation problems in arbitrary dimensions using eigenfunctions of an associated Hilbert-Schmidt operator. Computationally efficient implementations of these eigenfunction methods are derived in Chapter 5, and iterative methods involving them are discussed, although no contribution is made on that front.

Returning to the multiphysics setting, Chapter 6 develops a new technique for reducing the error and easing the logistical complexity associated with coupling multiphysics systems together. An example is discussed in Section 6.3.1 involving the new kernel-based approximation techniques of Chapter 4, the preconditioning techniques of Chapter 2, and the split finite differencing approach of Chapter

3. Beyond the approximation realm, Gaussian eigenfunctions are used to solve boundary value problems in Chapter 7, both through collocation and the Method of Particular Solutions. Chapter 8 explores statistical approaches for determining effective kernel parameterizations to take advantage of the newly found stable basis.

1.1 Multiphysics simulations

Because of the various communities involved in multiphysics projects and the contributions of a wide range of areas it is difficult to isolate specific problems which are emblematic of all multiphysics concerns. Select examples are discussed in this thesis, but a comprehensive review of this topic can be found in [106].

We cover key points of that paper now to help introduce the general structure of multiphysics systems; a more specific description is presented in Chapter 6, focusing on the coupling between the components. The simplest system which we shall consider "multiphysics" is a coupled equilibrium system

$$F_1(\boldsymbol{u}_1, \boldsymbol{u}_2) = 0,$$
$$F_2(\boldsymbol{u}_1, \boldsymbol{u}_2) = 0,$$

in which functions F_1 and F_2 represent components 1 and 2 with solutions $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ respectively. Because both models depend on each other, the solutions must be found simultaneously. A problem involving time can be discretized to yield this system (although other solution methods are available), so we limit our concerns to problems of the coupled equilibrium form. Also note that both F_1 and F_2 could themselves be multiphysics simulations, allowing for more than two coupled systems to still be described in this framework.

The functions F_1 and F_2 should be thought of as the component residuals, and, if so inclined, we could write a full system residual in the form

$$F(\boldsymbol{u}_1, \boldsymbol{u}_2) = \begin{pmatrix} F_1(\boldsymbol{u}_1, \boldsymbol{u}_2) \\ F_2(\boldsymbol{u}_1, \boldsymbol{u}_2) \end{pmatrix}.$$

This notation emphasizes the point that the system should be solved simultaneously, although Chapter 2 discusses the value of understanding the individual components within the full system solve.

Many applications currently use a Gauss-Seidel, or nonlinear Schwarz, solution approach, where each component is solved in an alternating fashion until convergence to the coupled solution is attained [156, 172]. While this technique is certainly the simplest method for coupling two components in a multiphysics simulation, researchers have found this technique causes problems in many circumstances [47, 170, 53, 54]. To prevent the instabilities which may accompany the nonlinear Gauss-Seidel, we choose to solve the fully coupled nonlinear system simultaneously. We use the Jacobian-free Newton Krylov method for this, which is introduced in Section 1.2.

The remainder of this section is spent briefly discussing two examples of multiphysics simulations which are used later in this thesis. In Chapter 2, the main example studied is magnetic confinement fusion within a tokamak [162]. The tokamak is a type of magnetic confinement fusion reactor. Despite the fact that this simulation consists of Boltzmann's equation solved throughout the domain, simplifying assumptions are made which break the problem apart into two regions: the edge and the core. Some work involves also separately simulating the material wall [36], which would produce a profile similar to Figure 1.1.

Ignoring the physics assumptions which must be made to reduce the standard

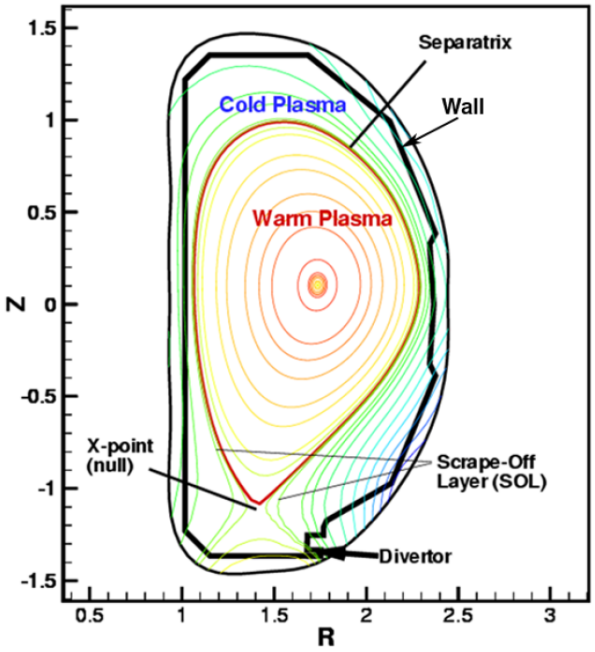


Figure 1.1: A plasma temperature profile for a core-edge-wall simulation within a tokamak reactor [106].

gyrokinetic model to this coupled MHD model, there are also computational techniques which must be developed to facilitate such a splitting in a high performance environment. Presently, research is underway to surpass the complications of a coupled core-edge-wall simulation [35], but work presented here focuses more intently on just the edge section. An explicit description of the relevant plasma transport equations is provided in Section 2.2.1.

We approach the edge region with the multiphysics approach described earlier - within the edge region is the coupled transport of plasma and neutral gases. Some transport codes traditionally consider only plasma terms due to ill-conditioning which arises when the coupled system is simulated. One of the topics we will analyze here is: How can knowledge of these components (the plasmas and neutral gases) be leveraged to improve the condition of the system, and thus the efficiency of a simulation? Chapter 2 considers the two components of the simula-

tion individually to expose key physical aspects, and then incorporate those into a preconditioning scheme.

Another multiphysics model, which is much further removed from a practical application but still of great use, is the linear critical gradient model [99]. In this model, a diffusion is simulated with nonlinear diffusivity which varies with the gradient of the temperature; depending on the choice of source for the problem, this can produce a pedestal similarly to what occurs at the core-edge interface of a tokamak [38]. This model is introduced in further detail in Chapter 3, and discussed in the coupling context in Chapter 6. We consider its solution using a new kernel-based technique in Chapter 7.

1.2 Jacobian-free Newton-Krylov

The method of spatial and time discretizations will not be analyzed here, as each application has its own assumptions and limitations which make certain choices more appropriate. In Chapter 7 we introduce a new technique for spatially discretizing PDEs, but outside of that section we will assume that the decision has been previously made by researchers in the various communities. Possible spatial discretizations are covered in various books including [116, 97], and specific time discretizations will be discussed as needed in Section 2.3.2 and Section 7.3.4. The end result of discretization is often a nonlinear system of the form

$$F(\boldsymbol{u}_k) = 0,$$

which has the solution $\boldsymbol{u}_k$ at the time t_k; this structure for multiphysics problems is described in Section 1.1. At times, fixed point iteration [107] can be used to solve this system, but the preferred approach in this work is Newton's method

[105] because of its potential for quadratic convergence.

Labeling $\boldsymbol{u}_k$ the unknown at the k^{th} time step, we can write this as

$$\text{For } n = 0, 1, ...$$

$$\mathsf{J}(F)(\boldsymbol{u}_k^n)\Delta\boldsymbol{u}_k^{n+1} = -F(\boldsymbol{u}_k^n) \tag{1.1a}$$

$$\boldsymbol{u}_k^{n+1} = \boldsymbol{u}_k^n + \Delta\boldsymbol{u}_k^{n+1} \tag{1.1b}$$

where $\boldsymbol{u}_k^n$ is the result of the n^{th} iteration of this scheme and $\mathsf{J}(F)(\boldsymbol{u})$ is the Jacobian of F evaluated at $\boldsymbol{u}$. Once convergence is declared (discussed in Section 1.2.1) at step N, the next time step is set as $\boldsymbol{u}_k = \boldsymbol{u}_k^N$. The initial guess $\boldsymbol{u}_k^0$ must be provided to the algorithm and the standard choice is the previous time step $\boldsymbol{u}_k^0 = \boldsymbol{u}_{k-1}$. The first initial guess $\boldsymbol{u}_1^0$ is the initial profile of the solution.

The usual practical barriers to the applicability of Newton's method include:

- Lack of an initial guess - Newton's method only converges quadratically with a sufficiently close initial guess. Finding one can, at times, be difficult.
- Lack of a Jacobian - Using Newton's method to solve $F(\boldsymbol{u}) = 0$ requires solving (1.1a); therefore, not only is F needed, but also $\mathsf{J}(F)$. For most applications, that Jacobian is unavailable or prohibitively expensive.
- Inefficiency of the linear solve - Even assuming that (1.1a) can be solved, that could become the most expensive part of the Newton iteration. How can the linear solve be performed efficiently?

These issues are addressed by the Jacobian-free Newton-Krylov [109].

For practical applications, (1.1b) is not the actual update used at each Newton step. Instead, a line search [104] is used which damps the magnitude of $\Delta\boldsymbol{u}_k^{n+1}$

so as to prevent oscillatory convergence behavior common during nonlinear solves. This changes the update portion of the nonlinear solve from (1.1b) to

$$\boldsymbol{u}_k^{n+1} = \boldsymbol{u}_k^n + \alpha^{n+1}\Delta\boldsymbol{u}_k^{n+1} \tag{1.2}$$

where $0 < \alpha_{min} < \alpha^{n+1} \leq 1$ is chosen according to some scheme. Experiments in Chapter 2 use a cubic interpolation scheme described in [50]. By using line search, a previously infeasible initial guess can be used and oscillations which might slow or prevent convergence will be damped. Other globalization approaches include physics trust regions [109] and pseudotransient continuation [85], but they are not discussed here.

Now we consider practical techniques used to circumvent explicit formulation of the Jacobian. In [48] it was determined that the superlinear convergence associated with Newton methods is preserved even when (1.1a) is solved only approximately. This revelation opens nonlinear solvers to the possibility of solving (1.1a) with an iterative method, thus introducing the *Krylov* half into the term Newton-Krylov method. When this Newton-Krylov method is performed in the absence of the true Jacobian, it is often called a matrix-free or Jacobian-free Newton-Krylov (JFNK) solve.

The iterative method of choice in this thesis is GMRES [84], since $\mathsf{J}(F)$ is rarely symmetric or positive definite and $\mathsf{J}(F)^T$ is unavailable. For practical purposes $\mathsf{J}(F)$ is actually also unavailable as commonly values of $F(\boldsymbol{u})$ are computed in a black-box fashion. This prevents the exact solve of (1.1a), but leaves the possibility of solving the system iteratively if the matrix vector product $\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b}$ can be

performed. By using finite differences, $\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b}$ can be approximated as

$$F(\boldsymbol{u}+\delta\boldsymbol{b}) = F(\boldsymbol{u}) + \delta\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b} + O(\delta^2)\frac{\boldsymbol{b}}{\|\boldsymbol{b}\|},$$
$$\iff \mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b} \approx \frac{F(\boldsymbol{u}+\delta\boldsymbol{b}) - F(\boldsymbol{u})}{\delta}, \tag{1.3}$$

for sufficiently small values of δ. Issues involving the accuracy of this approach are discussed in Chapter 3.

These approximate Jacobian-vector products are used in place of the true products during GMRES, which allows for a linear solve without ever actually computing the full Jacobian. As is often the case, the efficiency of the iterative linear solver is subject to the condition of the linear system in question, and many applications of scientific interest have poorly conditioned systems. To speed these linear solves, a preconditioner [84] needs to be created, and this topic is covered in Chapter 2.

The operator used to develop a preconditioner is created via Jacobian coloring [45], which allows for an approximate Jacobian to be computed via finite differences using a reduced number of function evaluations based on the nonzero structure of the matrix. This technique helps to overcome the third barrier to Newton's method mentioned above.

When written out in algorithmic notation, Jacobian-free Newton-Krylov looks like **Algorithm 1**. Some of these steps are left intentionally vague, such as the measure of convergence, choice of iterative linear solver and the method for line search. These choices are discussed in Section 1.2.1.

While it is assumed that the iterative method is preconditioned, it is not explicitly stated above as that process can be encapsulated totally within the iterative solver. Conversely, it is possible for the preconditioner to span multiple Newton

Algorithm 1 Jacobian-free Newton-Krylov

Given F, $\boldsymbol{u}_0$
Choose n_{MAX} {Max nonlinear iterations}
$\boldsymbol{u} \leftarrow \boldsymbol{u}_0$
while $n = 1, 2, ..., n_{MAX}$ and **not** converged **do**
  $\boldsymbol{b} \leftarrow F(\boldsymbol{u})$
  Solve $\mathsf{A}\Delta\boldsymbol{u} = \boldsymbol{b}$ iteratively, where $\mathsf{A}\boldsymbol{b} = (F(\boldsymbol{u} + h\boldsymbol{b}) - F(\boldsymbol{u}))/h$
  Determine α via line search
  $\boldsymbol{u} \leftarrow \boldsymbol{u} - \alpha\Delta\boldsymbol{u}$
end while
return $\boldsymbol{u}$

iterations (if the Jacobian were not expected to vary greatly between consecutive u values) and so the production level Jacobian-free Newton-Krylov algorithm can actually be much more complicated than the relatively naïve concept shown above. Doing so is of value in some simulations, and the improvement in cost will be discussed in Section 2.6.

1.2.1 Convergence criteria

As with any approximation or iterative scheme implemented computationally, decisions must be made regarding termination. This section explains the technical details behind declaring convergence, as well as some of the parameters which remains constant throughout this thesis. In the algorithm described above, the Newton iteration is continued for n_{MAX} steps, with the caveat that the iteration has not yet converged. For a general Newton-Krylov solver, there are several possible convergence criteria:

- Absolute Norm: Convergence is declared at step $n < N$ if $\|F(\boldsymbol{u}^n)\| < \epsilon$. Unless otherwise noted, this will be the active convergence criterion, and $\epsilon = 10^{-10}$.

- Relative Norm: Convergence is declared at step $n < N$ if $\|F(\boldsymbol{u}^n)\|/\|F(\boldsymbol{u}^0)\| < \gamma$.
- Relative Step Size: Convergence is declared at step $n < N$ if $\|\Delta \boldsymbol{u}_k^{n+1}\|/\|\Delta \boldsymbol{u}_k^n\| < \beta$.

Unless otherwise noted, all norms are 2-norms (i.e., $\|\cdot\| \equiv \|\cdot\|_2$). If the nonlinear iteration reaches step N without satisfying the prescribed convergence requirements, the iteration is considered to have failed/diverged. There are other possible causes for divergence, including line search failure or divergence of the linear solve (1.1a).

The line search can fail if the α_{min} value is chosen too large to find an α which decreases the norm; often this is the fault of a poor linear solve producing an inappropriate $\Delta \boldsymbol{u}$. For our experiments, $\alpha_{min} = 10^{-4}$.

The linear solve declares convergence when either the absolute or relative tolerance of the norm of the residual is small enough. Both of these criteria is checked at every linear iteration, so either of them can signal convergence to the linear solver; in practice, convergence is almost always due to the relative tolerance, set at 10^{-5}. GMRES could declare divergence if too many iterations pass without convergence (10000) or if the norm of the solution is greater than some preset maximum (10^5). Additionally GMRES has parameters which are of little consequence but are listed here for completeness: classical Gram-Schmidt orthogonalization is used; the restart value is 30.

Using finite differences to approximate matrix-vector multiplication introduces a parameter δ, as seen in (1.3). The accuracy of the finite difference approximation must be balanced by the error introduced from cancelation in the numerator,

meaning that sending $\delta \to 0$ is not a viable choice. The choice of

$$\delta = \sqrt{\varepsilon_{\text{rel}}} \frac{\sqrt{1 + \|\boldsymbol{u}\|}}{\|\boldsymbol{b}\|}$$

is used in the experiments of Chapter 2, unless otherwise noted. This choice is motivated by [135]. ε_{rel} is the relative error present in conducting a function evaluation, which here is estimated to be 10^{-14}, slightly higher than machine precision. For speed, the $\|\boldsymbol{u}\|$ term noted above can be retained from previous computation rather than recomputed for each application of (1.3). We will reevaluate the validity of this choice in some experiments in Chapter 3.

1.3 Multiphysics coupling

Multiphysics systems have many possible forms [106], but for the study of coupling within this thesis we will focus on one form: two independent boundary value problems which are coupled through an interface. An example of such a system, where two 2D models are coupled through a shared 1D boundary is depicted in Figure 1.2.

For a multiphysics problem of this form, each model consists of three regions. These regions are defined by their governing equations and their dependence on other regions to produce a solution, and are encapsulated in Table 1.1.

- **Interior region** - This region is governed by the PDE and its boundary conditions, and can be determined without using data from the other model. In fact, the solution here is also independent of the interface region, which is what differentiates them from the coupling region.

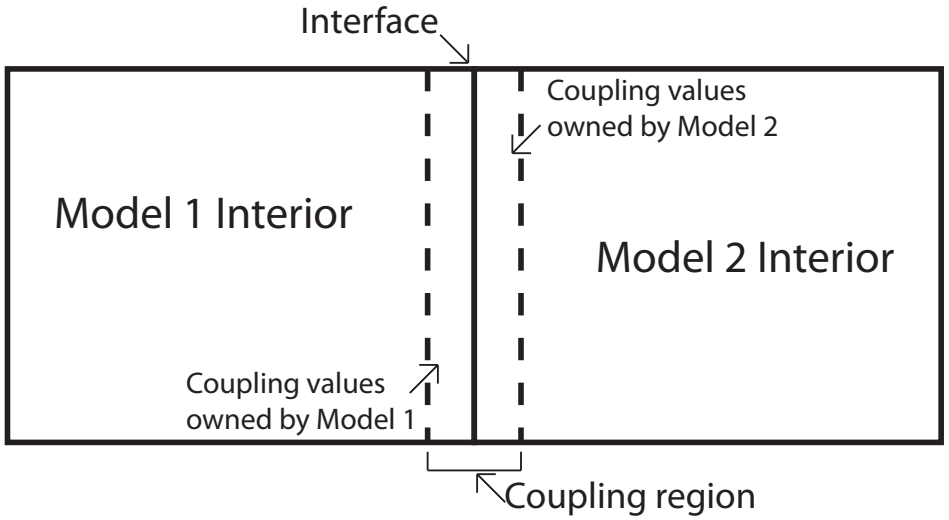


Figure 1.2: Two independent models are coupled through an interface. Each model has three distinct regions: the interior, coupling and interface regions.

- **Coupling region** - This region is governed by the PDE and its boundary conditions, and can be determined without using data from the other model. These values require data from both the local interior region and the local interface region.
- **Interface region** - This region is governed by the interface conditions, and requires values from the external model as well as the local model. These values require data from the local coupling region as well as the external coupling and interface regions.

		Solution components					
		$\boldsymbol{u}_1$	$\boldsymbol{u}_2$	$\boldsymbol{u}_1^C$	$\boldsymbol{u}_1^I$	$\boldsymbol{u}_2^C$	$\boldsymbol{u}_2^I$
Residuals	F_1	×		×			
	F_2		×			×	
	F_1^C	×		×	×		
	F_1^I			×	×	×	×
	F_2^C		×			×	×
	F_2^I			×	×	×	×

Table 1.1: This table notes which regions of the coupled multiphysics system are required to compute each residual. If a "×" symbol is absent, then those components can be ignored when computing that residual.

Two methods are commonly used to couple the simulations on these two domains: mortar methods and interpolation. Mortar element methods [15] involve creating an additional computational grid between the two existing grids. This introduces additional unknowns which play the role of boundary data for both subdomains. These methods have been applied to porous media [9], fluid-structure interaction [7], and other multiphysics and domain decomposition problems.

In contrast to mortar methods, interpolation methods use data present in one model to produce approximate boundary conditions for the other model. The effects of this approximation are difficult to analyze (error present in the input to the problem as well as computational error) and there are few results concerning the *a priori* convergence order of models coupled in this fashion. More solid footing can be found in *a posteriori* error estimation of multiphysics simulations [53, 33], but their implementation is beyond the scope of this thesis. Even with the meager error analysis, interpolation is a popular technique because of its simplicity, both in concept and in execution: take data that you have and use it to create data that you need.

The general form of a fully coupled multiphysics problem fitting the criteria described above is $F(\boldsymbol{u}) = 0$, as described in Section 1.1. Note that other multiphysics problems exist which cannot be written in this form; see [106] for a more thorough discussion. When written in terms of the regions described earlier, and using interpolation between domains rather than mortar methods which produce

additional unknowns, the solution vector and nonlinear residual function are

$$\boldsymbol{u} = \begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{u}_1^C \\ \boldsymbol{u}_1^I \\ \boldsymbol{u}_2^C \\ \boldsymbol{u}_2^I \end{pmatrix}, \qquad F(\boldsymbol{u}) = \begin{pmatrix} F_1(\boldsymbol{u}) \\ F_2(\boldsymbol{u}) \\ F_1^C(\boldsymbol{u}) \\ F_1^I(\boldsymbol{u}) \\ F_2^C(\boldsymbol{u}) \\ F_2^I(\boldsymbol{u}) \end{pmatrix}. \tag{1.4}$$

Here, $\boldsymbol{u}_1$, $\boldsymbol{u}_1^C$ and $\boldsymbol{u}_1^I$ are the solutions of model 1 in the interior, coupling and interface regions respectively. $\boldsymbol{u}_2$, $\boldsymbol{u}_2^C$ and $\boldsymbol{u}_2^I$ are the corresponding solution blocks for model 2.

Each of the function blocks has the same notation as the solution vector, e.g., $F_1^C(\boldsymbol{u})$ is the residual of the coupling region of model 1 and $F_2^I(\boldsymbol{u})$ is the residual of the interface region of model 2. As described in the itemized list above, F_1 and F_1^C are defined by the PDE in model 1, F_2 and F_2^C are defined by the PDE in model 2, and F_1^I and F_2^I are defined by the interface conditions. Notice that each of the regional functions are evaluated using the full solution vector $\boldsymbol{u}$ for conciseness; as shown in Table 1.1 each region only depends on part of $\boldsymbol{u}$.

Because the coupling and interior equations are defined by the PDE and boundary conditions, their discretization is left to the judgment of the application community — we want to study the effect of different discretizations of the interface conditions. Thus far little has been said about the interface; to ensure some level of consistency throughout the coupled model, the actual choice of interface conditions needs to be made by the scientists running the simulation.

Because it is impractical to consider all the possible coupling strategies in use today (see [106] for a survey) we restrict our study here to interface conditions

involving matched values and derivatives along the interface. Thus, the general form of interface conditions that we consider is

$$F^I(\boldsymbol{u}) = \sum_{k=0}^{r-1} a_{1,k}\mathcal{L}_k\boldsymbol{u}_1^I - \sum_{k=0}^{r-1} a_{2,k}\mathcal{L}_k\boldsymbol{u}_2^I = 0, \tag{1.5}$$

where r is the order of the PDEs, and $\mathcal{L}$ is some linear operator involving the derivatives (such as $\mathcal{L}_k(\boldsymbol{u}) = \boldsymbol{u}$, $\mathcal{L}_k(\boldsymbol{u}) = \frac{\partial}{\partial \boldsymbol{n}}\boldsymbol{u}$ or $\mathcal{L}_k(\boldsymbol{u}) = \nabla^2\boldsymbol{u}$). The coefficients $a_{1,k}$, and $a_{2,k}$ may depend on $\boldsymbol{x}$, and they define the relationship between the models in the interface conditions.

Interface conditions can be treated as boundary conditions, with similar terminology:

- **Dirichlet** : $a_{1,0} = a_{2,0} = 1$, all other $a_k = 0$, and $\mathcal{L}_0(\boldsymbol{u}) = \boldsymbol{u}$ is the identity. This produces the interface condition $\boldsymbol{u}_1^I = \boldsymbol{u}_2^I$ which matches solution values at the interface.
- **Neumann** : $a_{1,1} = a_{2,1} = 1$, all other $a_k = 0$, and $\mathcal{L}_1(\boldsymbol{u}) = \frac{\partial}{\partial \boldsymbol{n}}(\boldsymbol{u})$. This produces the interface condition $\frac{\partial}{\partial \boldsymbol{n}}\boldsymbol{u}_1^I = \frac{\partial}{\partial \boldsymbol{n}}\boldsymbol{u}_2^I$, that the normal derivatives are equal.
- **Robin** : $a_{1,0}, a_{2,0}, a_{1,1}, a_{2,1}$ potentially nonzero, $\mathcal{L}_0(\boldsymbol{u}) = \boldsymbol{u}$, and $\mathcal{L}_1(\boldsymbol{u}) = \frac{\partial}{\partial \boldsymbol{n}}(\boldsymbol{u})$. This produces the interface condition $a_{1,0}\boldsymbol{u}_1^I + a_{1,1}\frac{\partial}{\partial \boldsymbol{n}}\boldsymbol{u}_1^I = a_{2,0}\boldsymbol{u}_2^I + a_{2,2}\frac{\partial}{\partial \boldsymbol{n}}\boldsymbol{u}_2^I$, and may relate to a flux being conserved across the interface.
- **Laplace** : $a_{1,2} = a_{2,2} = 1$, all other $a_k = 0$, and $\mathcal{L}_2(\boldsymbol{u}) = \nabla^2\boldsymbol{u}$. This produces the interface condition $\nabla^2\boldsymbol{u}_1^I = \nabla^2\boldsymbol{u}_2^I$, and is more common in biharmonic, or other fourth order, problems.

Each model needs an interface condition, and the interface conditions must be independent for the problem to be well-posed [106], which explains why both F_1^I

and F_2^I are necessary. This general form of the interface conditions is made more concrete in Section 6.3.

The discretization of the $\mathcal{L}_k$ operators almost certainly involves values off the interface because of the presence of derivatives. The values required to approximate $\mathcal{L}_k$ on the interface define the coupling region. By choosing a specific interface discretization scheme, the coupling region is fixed to provide the necessary values. Conversely, a coupling region could be chosen and then a discretization could be built which best approximates $\mathcal{L}_k$ given that data. This relationship between choosing a coupling region and the quality of the resulting interface discretization is studied in the example in Section 6.3.1.

Solving the nonlinear system $F(\boldsymbol{u}) = 0$ with Newton's method involves a Jacobian $\mathsf{J}(F)(\boldsymbol{u})$ with the block sparsity structure

$$\begin{pmatrix} \mathsf{J}_1(F_1) & & \mathsf{J}_1^C(F_1) & & & \\ & \mathsf{J}_2(F_2) & & & \mathsf{J}_2^C(F_2) & \\ \mathsf{J}_1(F_1^C) & & \mathsf{J}_1^C(F_1^C) & \mathsf{J}_1^I(F_1^C) & & \\ & & \mathsf{J}_1^C(F_1^I) & \mathsf{J}_1^I(F_1^I) & \mathsf{J}_2^C(F_1^I) & \mathsf{J}_2^I(F_1^I) \\ & \mathsf{J}_2(F_2^C) & & & \mathsf{J}_2^C(F_2^C) & \mathsf{J}_2^I(F_2^C) \\ & & \mathsf{J}_1^C(F_2^I) & \mathsf{J}_1^I(F_2^I) & \mathsf{J}_2^C(F_2^I) & \mathsf{J}_2^I(F_2^I) \end{pmatrix}. \tag{1.6}$$

The notation used for the blocks is similar to the notation for the components of the residual, e.g., $\mathsf{J}_1^C(F_2^I)$ is the Jacobian with respect to the coupling variables of model 1 applied to the interface equations of model 2. Each of these blocks may also be sparse depending on the discretization of the components.

Even when using Jacobian-free Newton-Krylov, it is still useful to know the structure of the Jacobian to understand the cost of matrix vector products, and for approximating a full Jacobian using coloring. Note that the location of the

nonzero blocks corresponds to the dependencies described in Table 1.1. Also, all the Jacobians are evaluated with the same argument $\boldsymbol{u}$, so it is omitted to save space.

1.4 Kernel-based approximation methods

Section 1.3 discussed the coupling of multiphysics components, primarily facilitated through the transfer of data between component simulations. Error introduced in this transfer can be detrimental to the accuracy of the multiphysics simulation and therefore the transfer should be conducted as accurately as possible. This process is complicated by the potentially incompatible nature of the spatial discretizations in the component simulations. Our plan is to use kernel-based approximations to facilitate this coupling with high accuracy, but without the struggle of matching two disparate computational grids. Multiphysics coupling via kernel-based approximation is discussed in Chapter 6, but before we can address our new mechanism we must first introduce the use of kernel-based methods for scattered data approximation.

Many applications need to solve scattered data interpolation problems, where a set of N unique inputs $\{\boldsymbol{x}_k\}_{k=1}^N$ and associated outputs $\{y_k\}_{k=1}^N$ are used to approximate the function that generated that data [41]. The input data is assumed to be d-dimensional ($\boldsymbol{x}_k \in \mathbb{R}^d$), and the outputs should be scalars $y_k \in \mathbb{R}$; these restrictions are not necessary for more general problems in machine learning, but all the problems in this thesis are of this form. To solve scattered data interpolation problems, we assume that the underlying function of interest f can be approximated using a linear combination of N basis functions $\{\phi_k\}_{k=1}^N$. This approximation is

denoted by s,

$$s(\boldsymbol{x}) = \sum_{k=1}^{N} a_k \phi_k(\boldsymbol{x}),$$

and the coefficients a_k are determined by solving the interpolation equations

$$s(\boldsymbol{x}_k) = y_k, \qquad 1 \leq k \leq N.$$

This is a linear system of equations, which can be written as

$$\begin{pmatrix} \phi_1(\boldsymbol{x}_1) & \cdots & \phi_N(\boldsymbol{x}_1) \\ & \vdots & \\ \phi_1(\boldsymbol{x}_N) & \cdots & \phi_N(\boldsymbol{x}_N) \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}. \tag{1.7}$$

The choice of basis functions is significant both in the well-posedness of the problem and in the quality of the approximation. A common choice, especially for $d = 1$ (one dimensional) data, is to use polynomials. Even for this choice though there is flexibility, because a different choice of polynomial basis may allow for a more accurate interpolant by minimizing ill-conditioning during the solution process. For instance, the monomial basis

$$\phi_k(x) = x^{k-1}, \qquad 1 \leq k \leq N,$$

will produce a notoriously ill-conditioned Vandermonde matrix, while the Chebyshev basis

$$\phi_k(x) = \cos((k-1)\cos^{-1}(x)), \qquad 1 \leq k \leq N,$$

will produce the same result in exact arithmetic but with less ill-conditioning [168] when solving (1.7).

While uniqueness can be guaranteed for the interpolating polynomial in 1D for very basic conditions on $\{x_k\}_{k=1}^{N}$, the same cannot be said in higher dimensions.

This suggests that polynomials are not an appropriate basis for multidimensional interpolation; this idea was formalized in the Mairhuber-Curtis theorem on Haar spaces [121]. To ensure the existence of a unique interpolant s, we will implement a kernel-based interpolation scheme. Put simply, a kernel is any function of two variables $K \equiv K(\boldsymbol{x}, \boldsymbol{y})$, but for the purposes of this work, we will only consider kernels defined using radial basis functions,

$$K(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\|\boldsymbol{x} - \boldsymbol{y}\|).$$

Note that this restriction results in K always being symmetric with respect to its two arguments, i.e., $K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{y}, \boldsymbol{x})$. To further simplify the situation, we will be predominantly considering the Gaussian kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\varepsilon^2 \|\boldsymbol{x} - \boldsymbol{y}\|^2),$$

and, unless otherwise noted, this will be the function denoted by K. The value ε is a free parameter, often called a *shape parameter*, which determines the width of the Gaussians; small ε would produce Gaussians with very wide support, and large ε would produce Gaussians with very narrow support.

Because this kernel basis is a function of two variables, we have the opportunity to position, or orient, our basis functions in a useful manner. By choosing to position our positive definite basis functions at locations where data is provided, a unique interpolant must exist for arbitrary d. That means that the interpolant s must exist uniquely when

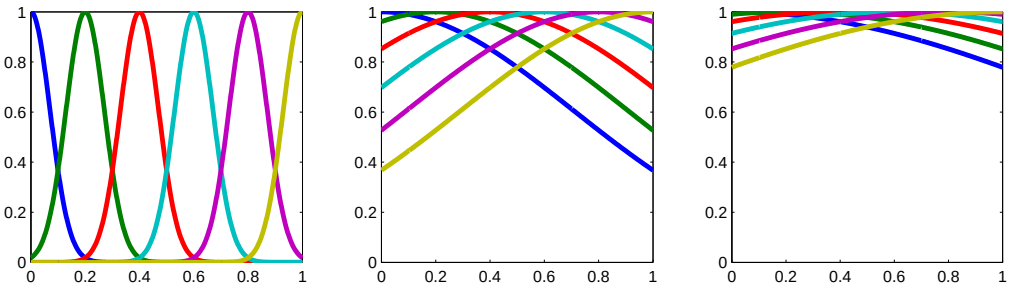
$$s(\boldsymbol{x}) = \sum_{k=1}^{N} a_k K(\boldsymbol{x}, \boldsymbol{x}_k). \tag{1.8}$$

This stems from both the fact that $\boldsymbol{x}_i \neq \boldsymbol{x}_j$ if $i \neq j$ and the fact that K is a positive definite kernel and therefore the matrix in (1.7) must be symmetric positive definite [60].

The freedom to interpolate scattered (unstructured) data without fear of a singular linear system or structural knowledge of the geometry of the data is one of the main benefits of kernel-based approximation; often times this approach is referred to as *meshfree* to emphasize this point. Applications in higher dimensions, including computer graphics [34], machine learning [137], data mining [95], and finance [126], use meshfree methods because of the impracticality associated with high dimensional structured sampling. Alleviating the curse of dimensionality is a powerful motivation for kernel-based schemes, but it is less significant in the multiphysics setting because most multiphysics problems occur in a relatively small number of dimensions. Instead, we are interested in the use of kernel-based schemes because their meshfree nature will allow us to connect individual components in a multiphysics simulation without worrying about the nature of their spatial discretizations.

Before we can consider the use of Gaussians to facilitate multiphysics coupling, we need to address a problem which has stymied the use of Gaussians and other very smooth kernels. Some very accurate choices of kernels, especially Gaussians, are associated with irrevocably ill-conditioned linear systems. This is caused by the *flattening* of the kernel, which allows it to more effectively include data in the interpolation, but at the cost of stability; see Figure 1.3 for an example.

In the past this has been termed the *uncertainty principle* [151], and it was believed by some to be intrinsic to very smooth kernels. Recent work has demonstrated that the interpolation problem is not ill-conditioned, but that the standard basis $\{K(\cdot, x_k)\}_{k=1}^{N}$ is an unstable basis for performing these computations. By writing our solution using a stable basis which spans the same space as the Gaussians, we can safely compute the interpolant. This approach has been explored for



(a) $\varepsilon = 10$, $\kappa = 1.07$ (b) $\varepsilon = 1$, $\kappa = 3.39 \times 10^6$ (c) $\varepsilon = .5$, $\kappa = 3.78 \times 10^9$

Figure 1.3: A sample set of $N = 6$ Gaussians is shown for three ε values, along with the condition number of the associated linear system. As ε decreases, the condition of the system increases.

problems on the sphere, using spherical harmonics as the stable basis [70].

Our approach uses an eigenfunction expansion of the Gaussians to produce a stable basis in $\mathbb{R}^d$; this is described in Chapter 4. After the mechanics of this stable approximation scheme have been established, Chapter 5 discusses a fast least squares solver for certain versions of this eigenfunction expansion. Its importance in reducing the computational cost is established, as well as its negative impact on stability. Iterative solvers for linear systems arising from the stable basis are also discussed, although no significant advances on this front have yet been performed.

Using this stable meshfree interpolation scheme, Chapter 6 demonstrates how function derivatives can be approximated with high accuracy. This infrastructure tool allows us to demonstrate how multiphysics simulations can be connected stably and accurately. Building on the success of the meshfree coupling, Chapter 7 adds another tool to our infrastructure by discussing the solution of boundary value problems using the stable basis. Chapter 8 studies methods for producing optimal kernel parameters, and analyzes the impact of the stable basis in successfully parameterizing kernels.

CHAPTER 2

PHYSICS PRECONDITIONING FOR EDGE PLASMAS IN TOKAMAKS

2.1 Introduction

Many systems of partial differential equations motivated by engineering and physics applications exhibit a wide range of temporal and spatial scales; increasingly these already complicated models are being coupled together for multiphysics simulations. Application-specific examples include compressible and reactive flow [31], porous media [42], fluid structure interaction [29], and fission power plants [78]. In each of these examples knowledge about the components of the system is used to produce a stable and efficient simulation that may provide new insights into physics or mathematical aspects of the models.

This chapter focuses on a numerical model that describes the transport of plasma and neutral species in the edge region of toroidal magnetic fusion energy devices. The strong external toroidal magnetic field ($\mathbf{B}$) in these devices results in a large transport anisotropy for the plasma with transport along the magnetic field (termed parallel) having a much faster timescale than transport across the magnetic field (termed perpendicular or radial). Note that in this chapter, parallel is used as an adjective in two different senses; one is transport along $\mathbf{B}$ just mentioned, and the second is using multi-processor parallel computational algorithms, and which meaning applies will be made clear from the context of a given section.

Modeling magnetically confined fusion in the presence of neutral gases is an ill-conditioned problem owing to the multiple time scales present, the complicated

geometry of the system and the competing physical effects of the various degrees of freedom. Coupling plasma and neutral transport together in a simulation produces a problem with multiple time scales which suffers, resulting in an ill-conditioned system, despite the fact that both components can be simulated effectively. This new difficulty arising from the joint simulation makes this application a suitable test case for building our multiphysics infrastructure.

The beginning of this chapter analyzes the contributions to the solution cost by the component terms under several simulation parameters including the time step Δt, parallel partitioning and solver design. Using insights gathered from this initial analysis, a physics-based preconditioner is constructed. This preconditioner shows improved solution time and scalability over simpler algebraic (such as ILU) and domain decomposition preconditioners. Finally, this approach to conducting the simulation is applied successfully to more complicated models with more degrees of freedom.

"Physics-based preconditioner" [129] is at times an ambiguous term, which we use here to describe an operator which uses more than just algebraic insight to improve the condition of a linear system. Some common purely algebraic preconditioners are discussed in Section 2.2.3 as well as [8]. Difficulty in physics-based preconditioning arises when determining how to incorporate physics knowledge into the operator. When this problem is handled successfully, the performance can benefit greatly, as was demonstrated for plasma transport in [37].

Our initial simulations motivated a split approach to preconditioning where variables with similar physical attributes are solved more cheaply. Each group of variables is then approximated and solved with different methods tailored to the physics of that group. This is in contrast to the approach of [37], where a semi-

implicit approximation was made to the fully implicit system which produced a preconditioner for the fully coupled system targeted to handle the different time scales with an easily invertible operator.

This chapter is based significantly on

M. McCourt, T. D. Rognlien, L. C. McInnes, H. Zhang, *Improving parallel scalability for edge plasma transport simulations with neutral gas species*, Computational Science & Discovery, 2012

which is cited as [124]. It is organized as follows. In Section 2.2 there is introductory information regarding the relevant plasma physics equations, the discretization used on those equations and the technique by which those discretized equations are solved. The plasma and neutral components of the system are analyzed in Section 2.3 through computations via PETSc [8]; insights are made regarding the viability of different solution schemes, and the infeasibility of existing preconditioners is inferred. In Section 2.4, a componentwise preconditioner (referred to as FieldSplit) is developed based on the study in the previous chapter, and is compared favorably to a global LU preconditioner. Finally, Section 2.5 explores the use of this Fieldsplit preconditioner on a variety of magnetic confinement simulations for which the existing preconditioners have had limited success. All experiments were conducted on the Fusion LCRC at Argonne National Laboratory.

2.1.1 A brief primer on computational plasma transport

The motivation for modeling transport in the edge region arises from a number of key questions that need to be answered, such as:

1. How is the exhaust plasma power that escapes from the core region distributed to material surfaces and what is its peak amplitude?
2. What is the rate of material erosion from plasma bombardment and what level of sputtered impurities get transported to the core plasma region?
3. How does exhausted tritium flow into and accumulate in materials?
4. How does helium ash from the fusion reaction get removed (pumped) at the periphery?
5. How does the core plasma get fueled by gas injection near the wall?
6. How does an edge energy transport barrier form?

The basic equation set used here for the plasma is a two-dimensional (2D), toroidally axisymmetric fluid model that describes the evolution of the ion density, parallel ion momentum, and separately, electron and ion temperatures. The neutral species are described by two possible fluid models, one where only particle transport is evolved, and the second where parallel neutral particle momentum is included. For either neutral model, the ion and neutral temperatures are assumed strongly coupled through collisions and are thus described by a common temperature. Both plasma and neutral equations have strong nonlinearities representing diffusive transport and coupled source/sink terms as will be shown explicitly in Section 2.2. It should be mentioned that there are also kinetic plasma transport models beginning to appear (XGC [38] and COGENT [51]) that include two added velocity space dimensions for describing the particle distribution functions. Also, kinetic Monte Carlo neutral codes (EIRENE [138] and DEGAS2[164]) are sometimes used, but these kinetic models are not considered here.

An early example of implementation of such an edge plasma model is the B2 code [24]. The numerical solver scheme used here was the SIMPLE algorithm [133]

used for codes that solve the Navier-Stokes neutral fluid equations. SIMPLE is an iterative algorithm that solves the equations in a certain order and uses a specific approximation to the pressure update to converge. More recently, B2 has evolved to SOLPS [20] that uses a partially implicit iterative scheme. The approach taken by UEDGE [142] was to solve the complete system of equations fully implicitly via a preconditioned Newton-Krylov method. Here a numerical finite-difference preconditioning Jacobian is formed and approximately inverted using ILUT [147]. All of these implementations were based on a single processor model where the variables of the global system were solved simultaneously. Subsequently, initial work was performed on a domain-decomposed parallel algorithm for the plasma equations where the preconditioner was solved independent on each subdomain with no overlap of information from the other subdomains [144].

2.1.2 Existing physics preconditioning techniques

The idea of using physics knowledge to improve the efficiency of a simulation is not new. Reactive transport simulations [90] have been performed where the preconditioner for the full Jacobian of the system is split into global transport and local reaction components. The same can be said for radiation diffusion systems [129]. Physics preconditioning in fluid dynamics codes has been successful [179] where the preconditioner served to accelerate a solution scheme already in place. Even within plasma physics, the concept of physics preconditioning has proven successful. In [37] the authors determined that they could improve the stability of simulating in multiple time scales by using a semi-implicit time stepping scheme to precondition the fully implicit solve. For a more exhaustive report of physics preconditioning ideas, refer to [106].

2.2 Physics equations and basic solver strategy

The basic equation set for the plasma is a two-dimensional (2D), toroidally axisymmetric fluid model that describes the evolution of the ion density, ion parallel momentum, electron temperature and ion temperature. The neutral species are described by two possible fluid models: one in which the neutral parallel velocity is computed by including only charge-exchange coupling to ions and the neutral pressure gradient, and the second in which neutral parallel inertia and viscosity are included. For either neutral model, the ion and neutral temperatures are assumed strongly coupled through charge-exchange collisions and are thus described by a common "ion" temperature. Both plasma and neutral equations have strong nonlinearities representing convective/diffusive transport and coupled source/sink terms, as shown in Section 2.2.1.

The UEDGE description of plasma transport here is not unique within the plasma physics community. Kinetic edge plasma transport models, including XGC [38] and COGENT [51], and kinetic Monte Carlo neutral codes, including EIRENE [138] and DEGAS2 [164], provide alternate methods of simulation with different advantages. The fluid transport model chosen in UEDGE more easily allows for longer time scale simulations to reach steady state.

2.2.1 Physics of the system

Edge plasma transport can be characterized via fluid moments of the underlying collisional kinetic equation for a plasma in a strong magnetic field; a general description is found in [25], or more recently [162]. Here studies are conducted using equations for plasma particle continuity, parallel momentum density, separate ion

and electron thermal densities, neutral particle continuity, and (at times) neutral particle momentum as described in [143]:

$$\frac{\partial}{\partial t}(n_{\mathrm{i}}) = -\nabla\cdot(n_{\mathrm{i}}\boldsymbol{v}_{\mathrm{i}}) + S_{\mathrm{i}}^{P} \tag{2.1a}$$

$$\begin{aligned}\frac{\partial}{\partial t}(n_{\mathrm{i}}m_{\mathrm{i}}v_{i\|}) = &-\nabla\cdot(n_{\mathrm{i}}m_{\mathrm{i}}v_{i\|}\boldsymbol{v}_{\mathrm{i}}) - \nabla_{\|}(n_{\mathrm{i}}T_{\mathrm{i}}) + en_{\mathrm{i}}E_{\|} - (\nabla\cdot\Pi_{\mathrm{i}})_{\|} \\ &- R_{\mathrm{i}\|} - m_i n_n \nu_{cx}(v_{i||} - v_{n||}) + S_{\mathrm{i}}^{m}\end{aligned} \tag{2.1b}$$

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{3}{2}n_{\mathrm{i,e}}T_{\mathrm{i,e}}\right) = &-\nabla\cdot\left(\frac{5}{2}n_{\mathrm{i,e}}T_{\mathrm{i,e}}\boldsymbol{v}_{\mathrm{i,e}}\right) + \boldsymbol{v}_{\mathrm{i,e}}\cdot\nabla(n_{\mathrm{i,e}}T_{\mathrm{i,e}}) - \nabla\cdot\boldsymbol{q}_{\mathrm{i,e}} \\ &- \Pi_{\mathrm{i,e}}\cdot\nabla\boldsymbol{v}_{\mathrm{i,e}} + Q_{\mathrm{i,e}} + S_{\mathrm{i,e}}^{E}\end{aligned} \tag{2.1c}$$

$$\frac{\partial}{\partial t}(n_{\mathrm{n}}) = -\nabla\cdot(n_{\mathrm{n}}\boldsymbol{v}_{\mathrm{n}}) + K_H^{\mathrm{i}} n_{\mathrm{e}} n_{\mathrm{n}} - K_H^{\mathrm{r}} n_{\mathrm{i}} n_{\mathrm{e}} \tag{2.1d}$$

$$\begin{aligned}\frac{\partial}{\partial t}(n_{\mathrm{n}}m_{\mathrm{n}}v_{n\|}) = &-\nabla\cdot(n_{\mathrm{n}}m_{\mathrm{n}}v_{n\|}\boldsymbol{v}_{\mathrm{n}}) - \nabla_{\|}(n_{\mathrm{n}}T_{\mathrm{n}}) - (\nabla\cdot\Pi_{\mathrm{n}})_{\|} \\ &+ m_i n_n \nu_{cx}(v_{i||} - v_{n||}) + S_{\mathrm{n}}^{m}.\end{aligned} \tag{2.1e}$$

Note that all terms except those with time derivative have been moved to the right-hand side of each equation, so that these use the same form as employed in the discussion of the time stepping in Section 2.3.2.

We analyze cases using deuterium (denoted D, having a proton-neutron nucleus with a single electron), typical of most present-day fusion devices. The primary variables evolved are n_i and n_n for D^+ and D^0 densities, $v_{i\|}$ and $v_{n\|}$ for ion and neutral velocities along $\mathbf{B}$, $T_i = T_n$ for the common ion/neutral temperature, and T_e for the electron temperature. Quasineutrality ($n_i = n_e$) is assumed. The parallel electric field, $E_{\|}$, can be replaced with plasma variables by using the electron parallel momentum equation, neglecting inertia and electrical current:

$$eE_{\|} = -T_e\nabla_{\|}(\ln n_e) - 1.71\nabla_{\|}(T_e), \tag{2.2}$$

where $n_e = n_i$ is the electron density in this quasi-neutral plasma.

Boundary conditions are applied to the second order differential set in Eqs. (1)

based on physical considerations. A more detailed discussion is available [143]. In summary, on the boundaries touching the hot core plasma or the side walls, either Dirichlet or Neumann boundary conditions can be applied to the densities and temperatures, specifying the variable value or its flux into or out of the simulation volume. Here we use Dirichlet conditions, and the numerical behavior is not sensitive to this choice. For the parallel velocities, on the core or outer wall, a radial slip condition is used, *i.e.*, zero normal derivative of the velocities. For the inner/outer divertor plate boundaries, the magnetic field lines intersect the material surfaces with strong plasma particle and energy flows onto the plates. The ion and electron temperature equations use mixed Robin boundary conditions to describe the flow of heat. The ion parallel velocity is set to the ion acoustic speed with the neutral parallel velocity being a fraction of that of the ion. A Neumann condition is applied to the ion density. The neutral flux into the plasma is proportional to the D^+ incident flux; that is, $\Gamma_{D^0} = -R_p \Gamma_{D^{+1}}$. This particle recycling is a major process at the divertor plates (and walls) and is characterized by R_p near unity.

The simplest neutral model neglects inertial and viscous terms in (2.1e), in which case $v_{n\parallel}$ can be computed algebraically from other variables, yielding a five-variable equation set used for some of the calculations in later sections. Physical terms and parameters include: S_i^p - source for neutral ionization and recombination, e - charge of an D^+ ion, $E_\parallel$ - electric field, $\Pi_{\mathrm{i,e}}$ - viscous force, $R_{\mathrm{i,e}\parallel}$ - friction force, $\boldsymbol{q}_{\mathrm{i,e}}$ - heat fluxes, $Q_{\mathrm{i,e}}$ - volume heating terms, $S_{\mathrm{i,e}}^E$ - external source terms, K_H^{i} - ionization rate coefficient, K_J^{r} - recombination rate coefficient. The term ν_{cx} is the ion/neutral charge-exchange rate. The parallel transport coefficients come from collisional theory [25], whereas the perpendicular transport coefficients arise from plasma turbulence and are typically deduced from comparison of midplane profiles with experimental data. Sample time scales present in an edge plasma

transport simulation are presented in Figure 2.1.

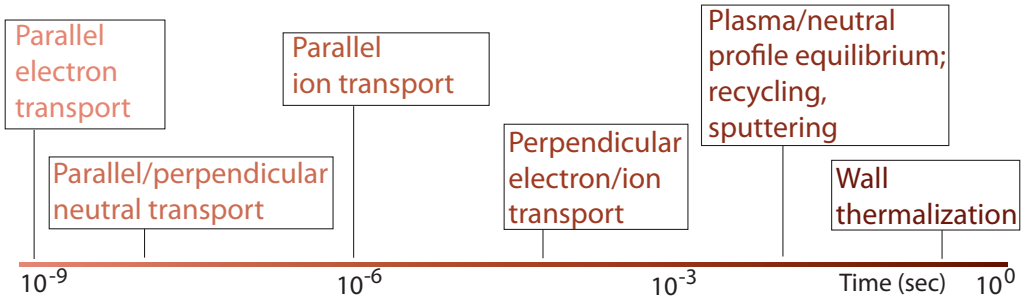


Figure 2.1: Significant phenomena in an edge simulation, and their associated time scales

The plasma transport is an anisotropic fluid flow problem, where the plasma *fluid* is charged and thus strongly affected by the external magnetic field. The plasma equations for density (2.1a), momentum (2.1b) and energy (2.1c) all govern variables which can be characterized as anisotropic fluids. The interaction between these quantities is significant in the poloidal direction, but much less so in the radial direction which contributes greatly to the magnetic confinement phenomenon. Indeed it is because of this that we see a significant anisotropy in the diffusion of plasma particles: a common ratio of effective diffusion constants for the plasma terms is

$$\frac{D_\perp}{D_\parallel} \sim \left(\frac{\rho}{\lambda}\right)^2$$

where $D_\perp$ is the effective diffusion coefficient in the radial direction, $D_\parallel$ is the effective diffusion coefficient in the poloidal direction, ρ is the gyroradius, and λ is the mean free path.

In contrast to the plasma terms, the neutral density governed by (2.1d) diffuses isotropically because neutral deuterium is unaffected by the external electric field.

This has a significant effect on the condition of the system because the physical discretization is designed in deference to the plasma terms and thus is extremely anisotropic. Section 2.3.3 will analyze this conditioning issue. The equations above describe the simplest transport simulation of interest involving both plasma and neutral gas terms; using this will allow physicists to study recycling and radioactivity within the material wall, among other phenomena.

Complexity can be added by removing the assumption that neutral inertial terms are negligible and thus increasing the number of unknowns to 6 by requiring the solution for the parallel neutral velocity (2.1e) within the system of differential equations. Alternatively, the neutral inertial term could be ignored and instead an impurity species of Neon gas can be added to the simulation. Electron impact excitation and ionization on Neon, as well as electron-ion recombination, results in a net energy loss to electrons via radiation loss (escaping photons). In the core region, such radiation degrades fusion power gain, but in the edge region, this radiative energy loss can be beneficial by distributing the plasma exhaust power over a wide area on the wall, thus minimizing the localized hot spots. Neon increases the number of independent fluid variables by 11 per cell because of the additional 10 Neon charge states (Ne^{+1} through Ne^{+10}) and one Neon gas species. Analysis of these simulations is presented in Section 2.5.

2.2.2 Domain discretization and decomposition

One of the complicating factors in simulating edge-plasma transport is the geometry. A quadrilateral-cell mesh is used for finite-volume discretization, with one coordinate being along magnetic flux surfaces and the other coordinate being orthogonal to the first or sometimes modified to fit along material surfaces; metric

coefficients preserve the original geometrical features in the logically rectangular domain. The finite-volume spatial discretization stencil contains 9 points: the central cell of interest and the 8 surrounding cells adjacent to the center. For each of the simulations we are running, the global grid sizes considered are production level; the exact grid size will be mentioned for each simulation.

In addition to the spatial discretization, we also discretize in time to convert the continuous transport equations to a fully discrete set of equations. For stability purposes, we choose the backward Euler method of time stepping. This discrete set of equations can now be fully encapsulated in the equation $F(u_k) = 0$, where u_k is the solution at time t_k. This concept of writing the full system as a single nonlinear residual problem was described in Section 1.1.

This thesis is not interested in the physical implications of the geometry or choice of boundary conditions, but it is necessary to consider the effectiveness of domain partitioning schemes on scalability. Consider the choice between a 1D (anisotropic) or a 2D (isotropic) decomposition posed in the right-most portion of Figure 2.2.

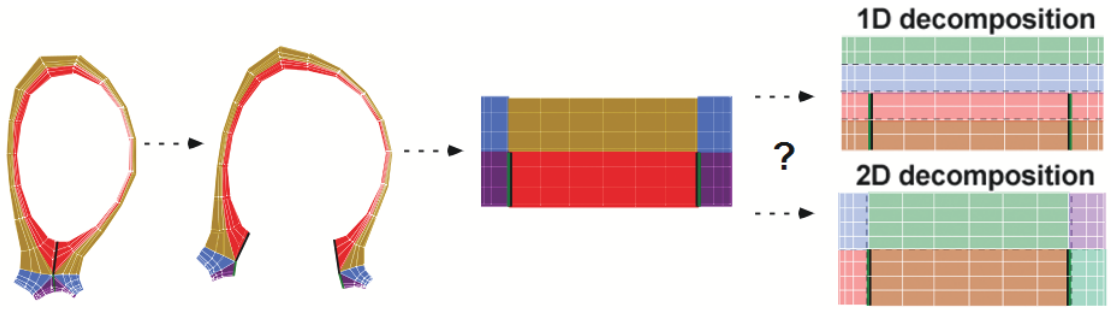


Figure 2.2: Different domain partitions are available depending on which cells should be more easily able to communicate with each other.

The 1D partitioning produces long strips with fewer necessary values in each partition; this is because all the neighboring values are needed, not just those

in the same radial strip. The 2D partitioning tries to accommodate this by including neighbors which are geometrically close in both the radial and poloidal directions. When considering the cost of parallel communication, occurring when data is passed between adjacent domains, the 2D decomposition scales better than the 1D variant because the domain surface-area-to-volume ratio is lower, resulting in less communication. This is confirmed for various processor counts in Figure 2.3, where 600 evaluations of the nonlinear residual (formed by discretizing in space and time) are computed with both partitioning strategies for an increasing number of processors..

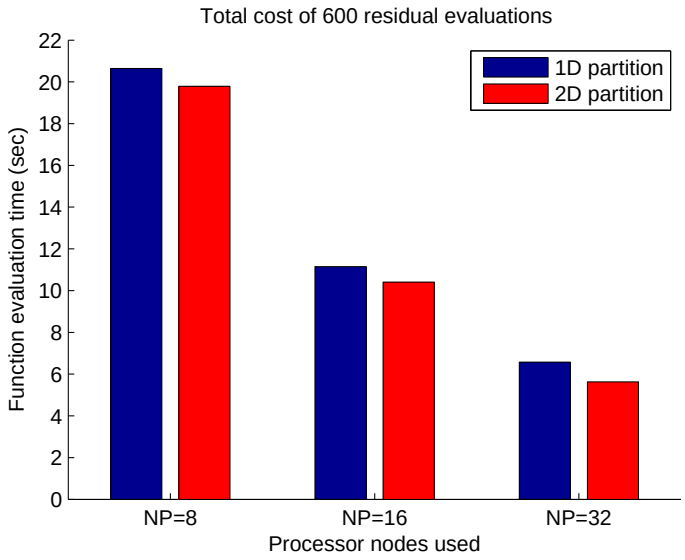


Figure 2.3: Parallel evaluations of $F(u_k)$ prefer the 2D partition because the volume to surface area ratio is lower, and thus less communication is required between domain partitions. 600 residual evaluations are used as a benchmark, without any problem specific significance.

When only 8 processors are used on the 256×128 mesh, there is little difference in the cost of function evaluations conducted on the 1D or 2D decomposition. Moving to more processors helps expose the different cost of communication between the choice of decompositions, with the 1D decomposition running more slowly. The choice of partition has a more complicated effect when the function evaluation is

considered within the context of the full solve; this will be discussed in Section 2.3.1.

Note that the presence of the branch cut in the rectangular domain prevents the use of a 2D decomposition in UEDGE with less than 6 processors. The 6 processor domain partition is shown in the bottom right of Figure 2.2. For a 2D decomposition, the first division of the domain occurs at the top of the branch cut to prevent a partition from having some of its terms crossing the branch cut and some not. The next two divisions occur on the branch cuts to simplify the communication. Because UEDGE enforces these requirements during domain decomposition, $NP = 6$ is the minimum for a 2D partition, and in fact $NP = 8$ will be the lowest processor node count for which a 2D partition is considered.

2.2.3 Preconditioner options and performance

PETSc provides a layered approach to solving PDEs consisting of time stepping tools, nonlinear solvers, linear solvers and preconditioners. Section 1.2 described the procedure for nonlinear solvers and linear solvers, but does not cover the topic of preconditioning. When we use the term preconditioning, we refer specifically to right preconditioning, which is the process by which a linear system $Ax = b$ is transformed into a new system $(AM^{-1})(Mx) = b$ with the same answer but a better condition number [84]. Left preconditioning is also useful in some circumstances, but we do not use it here.

Having a good condition number is significant because it allows iterative methods like GMRES to converge in fewer iterations. The ideal condition number is 1, which can be achieved by preconditioning with the true inverse $M^{-1} = A^{-1}$; this

is an unacceptable solution for our current problem both because the true matrix A is unavailable (recall the *approximate* matrix-vector product from Section 1.2.1) and because the exact inverse is prohibitively expensive.

In the absence of the true inverse, and in fact the true Jacobian, a preconditioner $M^{-1} \approx A^{-1}$ must be used which accurately solves as much of the spectrum as possible subject to the cost of applying it. The first approximation comes from the use of finite differences with coloring [45] to compute the matrix A which is used to represent $J(F)(u)$ during the linear solve. Recall that this finite difference approach only affects the computation of $J(F)$ and has no effect on F or the solution to $F(u) = 0$.

Now that an A has been produced, the question remains "How will M^{-1} be designed so that it resemble A^{-1} but can computed at less cost?" There is no unique technique which will effortlessly optimize computational cost, but here is a discussion of options which have found varying degrees of success for this problem:

- **LU** - Some problems may be so ill-conditioned that the fastest preconditioner is the true inverse A^{-1}. This method requires the most computational cost, but also the fewest linear iterations. Within this research, serial and parallel LU decomposition and solves are conducted with MUMPS [5].
- **ILU(k)** - If the LU preconditioner is too costly, the ILU(k) preconditioner [148] computes part of the LU decomposition. k is referred to as the level, and can be any nonnegative integer; this is roughly related to the additional nonzeros which may enter the matrix during factorization. This is often chosen small so as to minimize the memory allocation during factorization and the additional communication during solves.
- **ILUdt** - Like ILU(k), this also computes an incomplete LU decomposition;

however, rather than fixing a level of fill-in, this chooses to discard all nonzeros below some drop-tolerance. This is more difficult to apply effectively because the nonzero structure of the matrix cannot be determined prior to the factorization.

- **AMG** - Algebraic Multigrid [56] is a multilevel preconditioner with restriction and interpolation operators defined graph-theoretically. There are numerous tuning parameters available for AMG within the HYPRE [57] package which is used for experiments in this research. It should be noted that multigrid preconditioners are especially effective at solving elliptic problems.
- **ASM(n)** - When preconditioning in parallel, it is desirable to move as little data as possible between processor nodes. To accomplish this, the Additive Schwarz [161] preconditioner allows each processor to independently solve the portion of the domain it owns. This domain decomposition based preconditioner has a tuning parameter n which determines the overlap across subdomains to include in the solve. Also, within each subdomain a preconditioning strategy must be used, and often LU is chosen.

There are other preconditioners available, both within PETSc and in general, but the discussion in this section will be limited to those listed above for simplicity.

2.3 Preliminary Results

Having described the plasma transport simulation and the structure of the associated solver, we now analyze its efficiency for typical edge plasma parameters. The geometry corresponds to the DIII-D tokamak as shown in Figure 2.2. The initial conditions for these simulations are taken from a steady-state solution obtained for

core-boundary input parameters of fixed $T_e = T_i = 100$ eV and $n_i = 2.5 \times 10^{19}$ m^{-3}. The radial transport coefficients for all plasma variables are set to 1 m^2/s for simplicity, a value that is in the range typically deduced for present-day devices. The divertor plate particle recycling coefficient is set to $R_p = 0.9$. For the simulations presented in this section, the core-boundary values of T_e and T_i are then raised to 120 eV, and the new system is solved to a convergence level where the L_2-norm of the residual is reduced by a factor of 10^8 from the initial residual for a given time step.

2.3.1 Exploring the solver in the absence of neutral gases

Because the computational grid is laid out anisotropically in deference to the dominant plasma terms, there should be a well-conditioned problem when the plasma terms are handled separately and the neutral terms fixed. The timing results for "plasma only" simulation, involving (2.1a)-(2.1c), on a 256x128 grid are displayed in Table 2.1; the preconditioners being considered are the LU preconditioner and the ASM preconditioner with overlap 1 and LU on each block. Note that for $NP = 1$ these two preconditioners are actually the same since no domain decomposition is taking place.

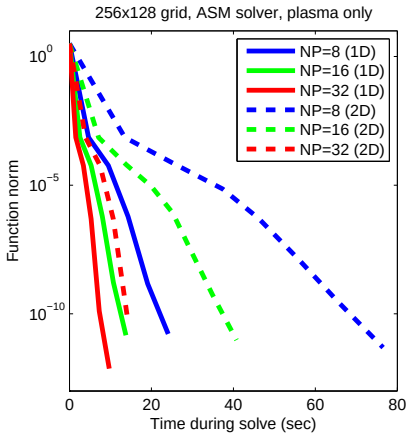
Processor Nodes	**Time (sec)**		**Speedup**	
	LU	ASM	LU	ASM
2	109	93	1.85	2.18
4	61	46	3.33	4.43
8	37	24	5.50	8.44
16	25	14	8.01	14.83
32	18	10	11.03	20.94

Table 2.1: Plasma terms are easily solved with domain decomposition methods

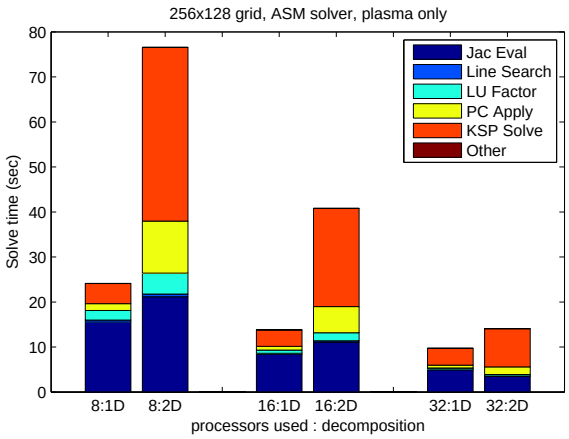
By choosing a domain decomposition which respects the physics of the plasma

transport, the ASM preconditioner is able to scale much more effectively than the LU preconditioner. ASM is less effective than LU on ill-conditioned systems so its success here shows that the "plasma terms only" problem is rather well conditioned with $\Delta t = 10^{-4}$.

The outstanding performance in Section 2.3 is a benefit of the domain decomposition, since the blocks selected by ASM are the same blocks chosen during domain partitioning. To determine the effect of the partitioning on the quality of the preconditioner, refer to Figure 2.4a which compares the solve times using both a 1D and 2D domain partition.



(a) 1D partitioned ASM solves consistently faster for plasmas



(b) Overall cost is greater in 2D than 1D

Figure 2.4: When using ASM on plasma only simulations, 1D partitioned solves are faster because nonlinear solve cost overwhelms the function evaluation savings in 2D.

Processor nodes	**Linear Iterations**	
	1D	2D
8	21	129
16	30	134
32	50	161

Table 2.2: 1D partitioned ASM requires fewer linear solve iterations.

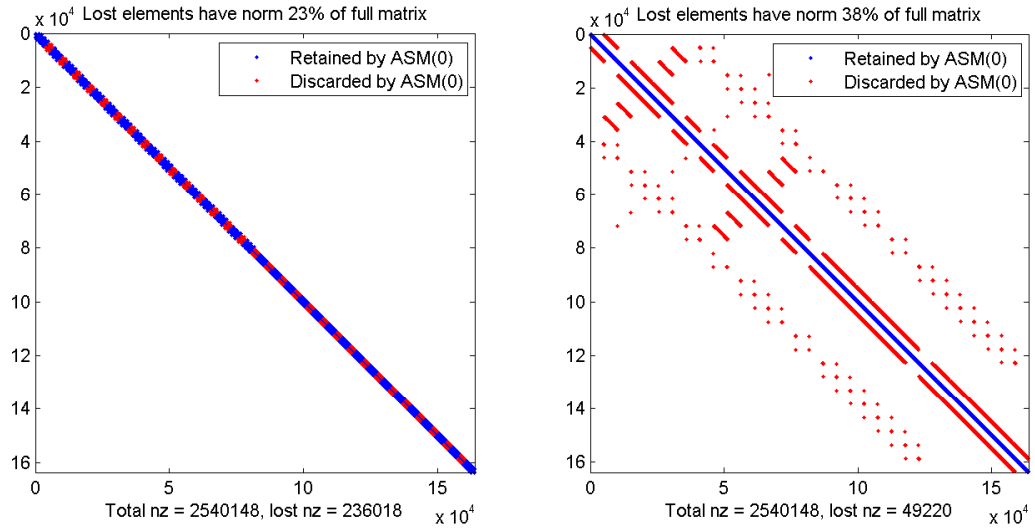
Clearly, Figure 2.4a indicates that the ASM preconditioner prefers a 1D over a

2D domain decomposition. This is supported by the physics of the system in which the plasma terms prefer to not cross magnetic lines and thus the most significant coupling for the plasma variables is in the poloidal direction and not the radial direction. The 1D domain decomposition most respects that by retaining all the couplings in each radial strip and then neglecting some of the couplings across radial strips. Because less information is lost when using the 1D decomposition (i.e., few terms cross the radial partition) the quality of the preconditioner remains high and the solve is conducted more quickly.

Figure 2.4b shows the proportion of time during the nonlinear solve spent in each of the components. The components listed here are

- **Jac Eval** - *Once per nonlinear iteration* - Evaluation of the approximate Jacobian used for preconditioning only,
- **Line Search** - *Once per nonlinear iteration* - Improves globalization of JFNK by reducing the size of the suggested Newton step (see Section 1.2),
- **LU Factor** - *Once per nonlinear iteration* - Production of triangular factors needed during preconditioning, but may not be full LU decomposition,
- **PC Apply** - *Once per linear iteration* - Preconditioning linear solve using factors from *LU Factor*,
- **KSP Solve** - *Once per linear iteration* - Jacobian-free matrix vector products and optimization required by **K**rylov **S**ubs**P**ace (KSP) methods, in this problem GMRES is used,
- **Other** - Convergence monitor updating, incidental memory allocations, other trivial costs.

If the quality and speed of the preconditioner were unchanged by the choice of partition, then the isotropic partition would consistently provide a faster solve. Figure 2.4b instead shows that despite the function evaluations scaling better in 2D than 1D, the speed of the solve is still much slower; this is because the 2D preconditioner is less effective, as can be seen in Table 2.2 where the needed number of KSP iterations is much greater for the 2D than 1D partition. The difference in linear iterations, and also computational time, associated with the ASM preconditioner is directly related to the difference in sparsity structure of the matrix, which can be seen in Figure 2.5.



(a) 1D nonzero pattern

(b) 2D nonzero pattern

Figure 2.5: The 1D and 2D decompositions produce different global matrix orderings

Even though there are more nonzeros in the off-diagonal blocks of the 1D decomposition, they are more easily recovered through small increases in the overlap of the Additive Schwarz (ASM) preconditioner. In contrast, the values lost in the 2D partition are very difficult to recover through increased overlap. This additional cost allows the 1D decomposition to be competitive with the 2D decomposition despite the greater communication cost during function evaluation.

2.3.2 Effect of time stepping

It was discussed in [85] that when problems of the form

$$\frac{du}{dt} - F(u) = 0,$$

are solved with Newton iteration, the associated linear systems will have improved condition with a smaller time step Δt. This can be seen for a simple implicit time discretization scheme

$$\frac{u_k - u_{k-1}}{\Delta t} - F(u_k) = 0$$

which then has the linearized system

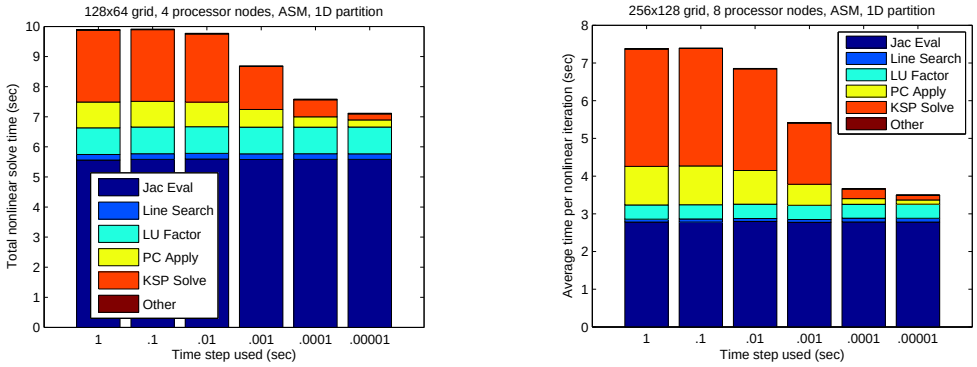
$$\underbrace{\left(I - \Delta t\, J(F)\left(u_k^{(n)}\right)\right)}_{=A} \Delta u_k^{(n)} = u_k^{(n)} - u_{k-1} - \Delta t\, F\left(u_k^{(n)}\right)$$

that needs to be solved iteratively. As is well known [169], an iterative system is best solved when the spectrum of the system matrix, in this case A, has eigenvalues clustered away from 0.

Assuming the Jacobian term $J(F)\left(u_k^{(n)}\right)$ is ill-conditioned (were it not this problem is easily solved) it has eigenvalues which are spread out over a large region of the complex plane. By multiplying these eigenvalues by a small Δt term the region they encompass is shrunk towards zero by a factor of Δt.

By itself this would not cause any difference in the condition because both the largest and smallest eigenvalues would be scaled by the same factor and thus the ratio of the two would remain unchanged. When A is actually formed, the identity matrix is added to the scaled Jacobian pushing these eigenvalues which have been scaled away from 0. As the $\Delta t \to 0$ the eigenvalues should approach a cluster around 1 on the real axis. Such a matrix is extremely well conditioned and solving such a system should therefore require little effort.

Figure 2.6 shows this effect on the ASM preconditioner. Because most of the values exchanged between domains are neglected by this preconditioner it is not appropriate for an extremely ill-conditioned system. Its ineffectiveness is apparent in Table 2.3b where, for $\Delta t = 1$, 78 linear iterations are required on average for each nonlinear iteration. That number decreases to 4 linear iterations on average for $\Delta t = 10^{-5}$ which in turn cuts the average nonlinear iteration time in half.



(a) Nonlinear solves are easier with smaller time step

(b) Linear solve proportion decreases with time step

Figure 2.6: Decreasing the time step improves the quality of ASM preconditioners

Time step	Total linear iterations
1	127
.1	126
.01	120
.001	78
.0001	34
.00001	15

(a) Total linear iterations decrease with the time step

Time step	Average linear iterations
1	78
.1	78
.01	67
.001	41
.0001	7
.00001	4

(b) Average linear iterations per nonlinear iteration decrease with time step

Table 2.3: Significantly fewer linear solves are required for smaller time steps.

As the quality of the preconditioner improves the number of linear iterations required per nonlinear iteration should decrease. This is noticeable in Figure 2.6a where the proportion of time spent on *Once per linear iteration* costs (**PC Apply** and **KSP Solve**) decreases significantly as Δt decreases.

A decrease in Δt may also change the number of nonlinear iterations in addition to the observed change in the cost of linear iteration per nonlinear iteration. This happens not because the preconditioner is better but because the initial guess is closer to the solution. For a time dependent problem it is often the case that the initial guess at time t_n, $u_n^{(0)}$, is chosen to be the solution from the previous time step u_{n-1}. As a result, choosing a smaller time step provides an initial guess closer to the solution because the solution should change less over the shorter period of time. The quality of the preconditioner is independent of the number of nonlinear iterations (assuming the linear solves converge) and thus the *Once per nonlinear iteration* costs should not change with the time step.

Although it has been determined that the time for a nonlinear solve decreases with the time step, it has not been concluded that doing so is more efficient. If our goal is to reach a final time of $T = .1$ then it may be more efficient to take one very slow step of $\Delta t = .1$ than to take 1000 faster steps of $\Delta t = 10^{-4}$. Because the relationship between Δt and total solve time is too complex to study analytically, computational analysis is appropriate. In Figure 2.6 the total solve time is obviously decreasing, but it would have to be decreasing exponentially to compensate for the significantly smaller time steps.

For smaller time steps, the increased number of nonlinear solves would overwhelm the shorter solve time, indicating that in general there may be an optimal time step for a given simulation and preconditioner at which the computational cost per simulation second is minimized. Additionally, it may be possible to lag Jacobian computations and reuse Jacobians across time steps, thereby increasing the efficiency of smaller time steps. While a time step $\Delta t \in [10^{-4}, 10^{-3}]$ is appropriate for a coupled core-edge simulation, steady state problems have a final time

$T = O(1)$. These ideas allow for a steady state simulation to potentially be solved more quickly with a smaller time step, although actual experiments involving Jacobian lag will not be covered here.

2.3.3 Result for neutral components with fixed plasma background

The plasma transport can be thought of as having two components - ionized plasmas and neutral gases. Because those two components have different physical properties, it seems prudent to understand each of them individually before drawing any conclusions about their coupling within the simulation.

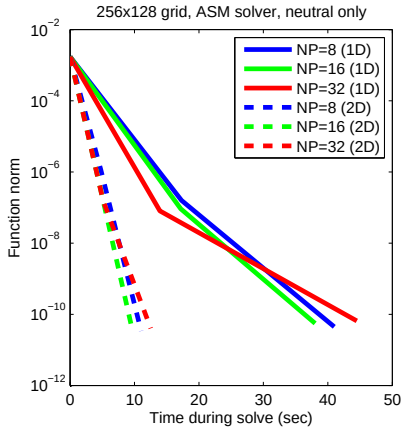
To gain the same insight on neutral gas terms in the absence of plasma terms, the experiments from Section 2.3.1 should be repeated. There are small technical complications, namely the fact that a 2D decomposition cannot be performed on any less than 6 processors; refer back to Section 2.2.2 for a discussion of this. Within Table 2.4, this causes experiments with less than 8 processors ($NP < 8$) to be on a 1D decomposition despite larger experiments using a 2D decomposition.

Processor Nodes	**Time (sec)**		**Speedup**	
	LU	ASM	LU	ASM
2	6.31	42.6	2.04	0.30
4	3.30	38.7	3.91	0.33
8	1.81	11.0	7.14	1.17
16	1.10	9.67	11.8	1.33
32	0.80	12.53	16.21	1.03

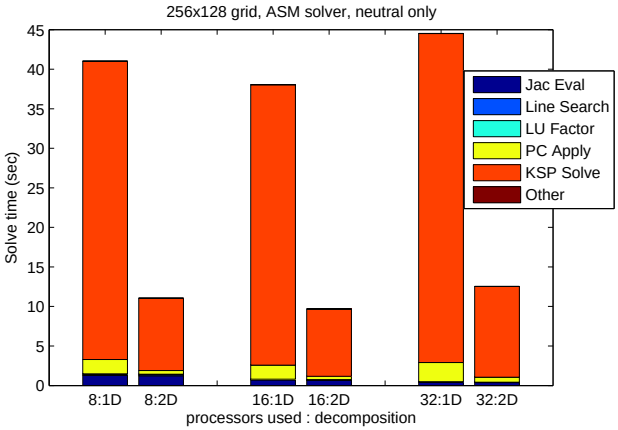
Table 2.4: Neutral terms are poorly solved with ASM preconditioner

Table 2.4 shows that there is significantly better performance for the LU preconditioner over ASM, even for the 2D domain decomposition which would logically

be preferred by the neutral terms as they diffuse isotropically. To confirm that 2D is the proper choice of partition, both 1D and 2D partitions are tested for 8 and 16 processors in Figure 2.7a. There it is apparent that the best domain decomposition is the 2D decomposition, but it was unable to scale as well as the LU preconditioner from Table 2.4.



(a) 2D partitioned ASM solves consistently faster for neutrals



(b) Overall cost is greater in 1D than 2D

Figure 2.7: With only neutrals, 2D ASM is superior to 1D ASM, but both are inferior to full LU.

Processor nodes	**Linear Iterations**	
	1D	2D
8	454	114
16	782	200
32	1498	477

Table 2.5: 2D partitioned ASM requires fewer linear iterations.

The main point of these results is to note that for the neutral deuterium species, the preferred preconditioner seems to be LU regardless of the choice of decomposition; this is in contrast to the results from Section 2.3.1 for the plasma terms only. One of the contributing factors to this surprising occurrence may be the size of the problem under review: there are only 32768 unknowns for the neutral only problem which may be too small for the cost of fill-in and communication associated with

the LU preconditioner to outweigh the savings in fewer linear iterations. This idea is supported by Table 2.5 which notes that ASM is consistently taking hundreds of linear iterations per nonlinear iteration - Figure 2.7b shows that the time spent in **KSP Solve** is nearly the entire cost of the nonlinear solver.

Another possibility is that the condition of the neutral only problem is so severe that solving it iteratively becomes prohibitively expensive. A condition estimate [92] of the neutral only system shows $\kappa = 1.8 \times 10^{11}$ whereas the plasma only system, which is 4 times larger, has condition estimate $\kappa = 3.1 \times 10^{7}$. One possible cause of this ill-conditioning is discussed in the Section 2.3.4, but regardless of the cause, this significant disparity would surely contribute to the contrasting performance of ASM for the "plasma only" and "neutral only" simulations.

2.3.4 Analysis of neutrals on an anisotropic mesh

To understand the impact of adding the neutral component on the parallel scaling, we first consider the properties of the neutral continuity equation. For a high-density plasma typical of the edge plasma region, the neutral flow velocity is determined largely by the pressure gradient and charge-exchange friction terms in (2.1e). Thus,

$$\mathbf{v}_n \approx \mathbf{v}_i - \nabla(n_n T_n/m_n)/(n_n \nu_{cx}),$$

where $T_n = T_i$ is the approximate neutral temperature and ν_{cx} is the charge-exchange collision frequency. Consequently, the neutral particle continuity equation (2.1d) has the form of a simple isotropic diffusion equation.

For the purposes of this example, we consider a simple rectangular grid shown in Figure 2.8 with $\Delta y = \alpha \Delta x$ for $0 < \alpha \ll 1$. The governing PDE is the Poisson

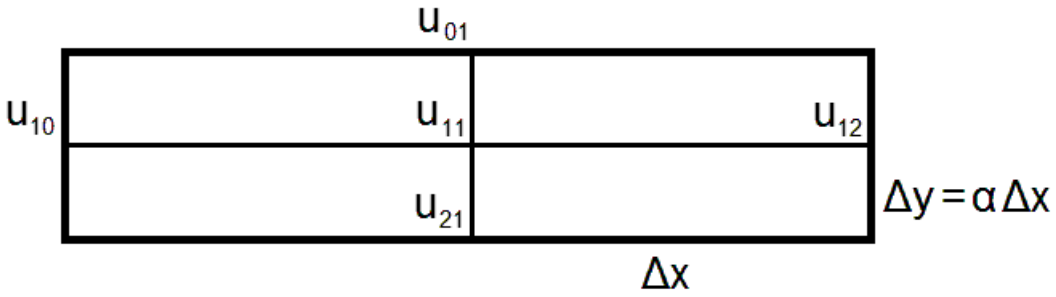


Figure 2.8: Grid spacing for Δx is α times greater than Δy

equation with homogeneous Dirichlet boundary conditions

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u(x,y) = f(x,y) \qquad \text{Interior,}$$

$$u(x,y) = 0 \qquad \text{Boundary.}$$

Using finite differences to discretize the PDE onto the anisotropic grid from Figure 2.8 produces a system of equations

$$\frac{-2u_{ij} + u_{i+1\,j} + u_{i-1\,j}}{(\Delta x)^2} + \frac{-2u_{ij} + u_{ij+1} + u_{ij-1}}{(\Delta y)^2} = f_{ij} \qquad i = j = 1$$

$$u_{ij} = 0 \qquad \text{else}$$

where f_{11} is f evaluated at the same location as u_{11}. Because of the structure of this particular stencil, the corner points do not affect the interior point, so their value need not be included in the solution of the system. Their omission leaves a system of 5 unknowns which after substituting $\Delta y = \alpha \Delta x$ takes the $Ax = b$ form

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ \alpha^2 & 1 & 1 & \alpha^2 & -(2+2\alpha^2) \end{pmatrix} \begin{pmatrix} u_{01} \\ u_{10} \\ u_{12} \\ u_{21} \\ u_{11} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \alpha^2(\Delta x)^2 f_{11} \end{pmatrix}$$

This particular system can be easily inverted to the form $x = A^{-1}b$

$$\begin{pmatrix} u_{01} \\ u_{10} \\ u_{12} \\ u_{21} \\ u_{11} \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ \frac{\alpha^2}{2+2\alpha^2} & \frac{1}{2+2\alpha^2} & \frac{1}{2+2\alpha^2} & \frac{\alpha^2}{2+2\alpha^2} & \frac{-1}{2+2\alpha^2} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \alpha^2(\Delta x)^2 f_{11} \end{pmatrix}$$

giving the 1-norm condition number of the system as

$$\begin{aligned} \kappa_1(A) &= \|A\|_1 \|A^{-1}\|_1 \\ &= (2 + 2\alpha^2)\left(1 + \frac{1}{2 + 2\alpha^2}\right) \\ &= 3 + 2\alpha^2 \end{aligned}$$

Since the restriction was made that $0 < \alpha \ll 1$, $\kappa_1(A) \in (3, 5]$, which is a well-conditioned system. Initially that may seem like a positive outcome, but the situation muddies when the implications of that assumption on the right hand are analyzed. As $\alpha \to 0$, $\alpha^2(\Delta x)^2 f_1 1 \to 0$ as well, which leaves a right hand side which begins to look homogeneous as the anisotropic grid becomes more extreme. To compensate for this shrinking nonzero term the matrix condition number should increase, but instead it is bounded from above by 5 and thus the system becomes insensitive to the very small terms.

It is tempting to think then that this issue is simply a function of the choice of $\Delta y = \alpha \Delta x$ for $0 < \alpha \ll 1$ rather than $\Delta x = \gamma \Delta y$ for $\gamma \gg 1$. Reformulating the system with this discretization as $\hat{A}$ produces a very similar matrix and a similarly structured condition number

$$\begin{aligned} \kappa_1(\hat{A}) &= (2 + 2\gamma^2)\left(1 + \frac{\gamma^2}{2 + 2\gamma^2}\right) \\ &= 2 + 3\gamma^2 \end{aligned}$$

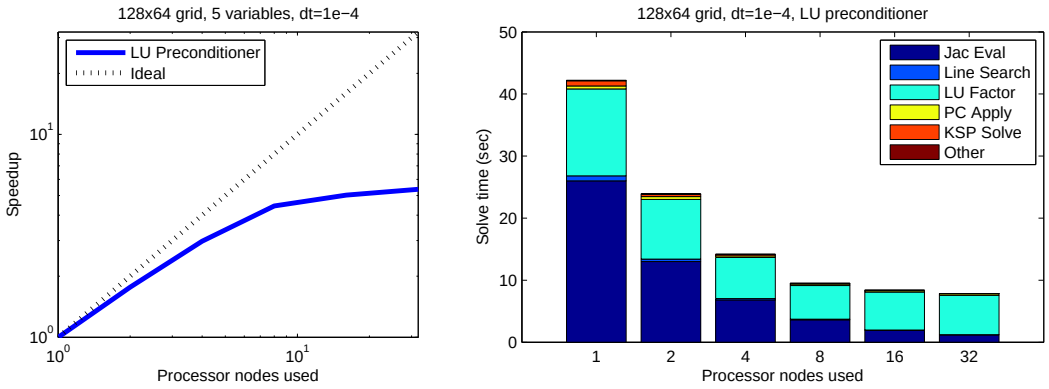
Unfortunately, an anistoropic grid is characterized in this formulation as $\gamma \to \infty$, which shows not only is the condition number not bounded for this system, but it grows quadratically with γ. Thus, in this more traditional sense, the system is rather ill-conditioned as a direct result of the isotropic diffusion on the anisotropic grid.

2.3.5 Coupled plasma and neutral solver results

Initial experiments involving UEDGE [142] were attempts to study the steady state behavior of the simulation. The time step of choice was $\Delta t = 1$, after one step of which the plasma transport simulation is in its steady state. Unfortunately, using such a large time step produced a linear system so ill-conditioned that it could only be solved using the full LU factorization as the preconditioner. Choosing a simpler preconditioner than LU, such as incomplete LU (ILU(k)) or SSOR, proved ineffective as did the more complicated algebraic multigrid (AMG): the linear solves failed to converge in each of those cases. ILUdt was only successful with a very low drop tolerance, to the point that the cost is comparable to the LU preconditioner.

One of our significant goals is to run UEDGE scalably on multiprocessor machines, and for this purpose the LU preconditioner is limited. As has been documented [84], the full LU decomposition loses scalability because of the additional nonzeros generated during the factorization which change the nonzero structure of the factored matrices. Figure 2.9 shows this lack of strong scalability when we solve the fully coupled plasma/neutral simulation involving (2.1a)-(2.1d).

In Figure 2.9a there is a clear dropoff in performance of the solver as the number of processor nodes in the computation increases. Figure 2.9b analyzes this result



(a) Loss of scalability is significant (b) Time spent factoring the matrix stops scaling

Figure 2.9: A LU preconditioner shows limited strong scalability for this problem

and shows that the time required for performing the LU factorization fails to scale as the number of processors increases. To understand Figure 2.9b it is important to note that for $NP = 1$ all the components of the solve are visible. Each of the components either scales well (**Jac Eval**) or becomes neglible compared to the cost of the solve (e.g., **Line Search**) except the **LU Factor** bar. That bar stops decreasing in size for $NP > 4$ and in fact grows as more processor nodes are used - this means that even as more processors are used in the computation, more total time is spent factoring the preconditioner.

This troubling performance indicates that the choice of preconditioner is a significant concern for scalability, and also that the LU preconditioner may not be ideal for larger problems. Because other preconditioners have failed under these same conditions, we must consider a new preconditioner which can solve the coupled problem without the computational limitations of the global LU preconditioner.

2.4 A field-split preconditioner for improved plasma/neutral performance

Section 2.3.5 concluded with the belief that solving the coupled neutral and plasma species simultaneously on an anisotropic grid is too ill-conditioned for a domain decomposition based preconditioner such as ASM to succeed efficiently. This is in fact an oversimplification, since with some large amount of overlap between domains preserved in the preconditioner the full LU would be recovered from ASM; doing so would however negate the reduction in communication desired by using ASM and thus is no better than the LU factorization.

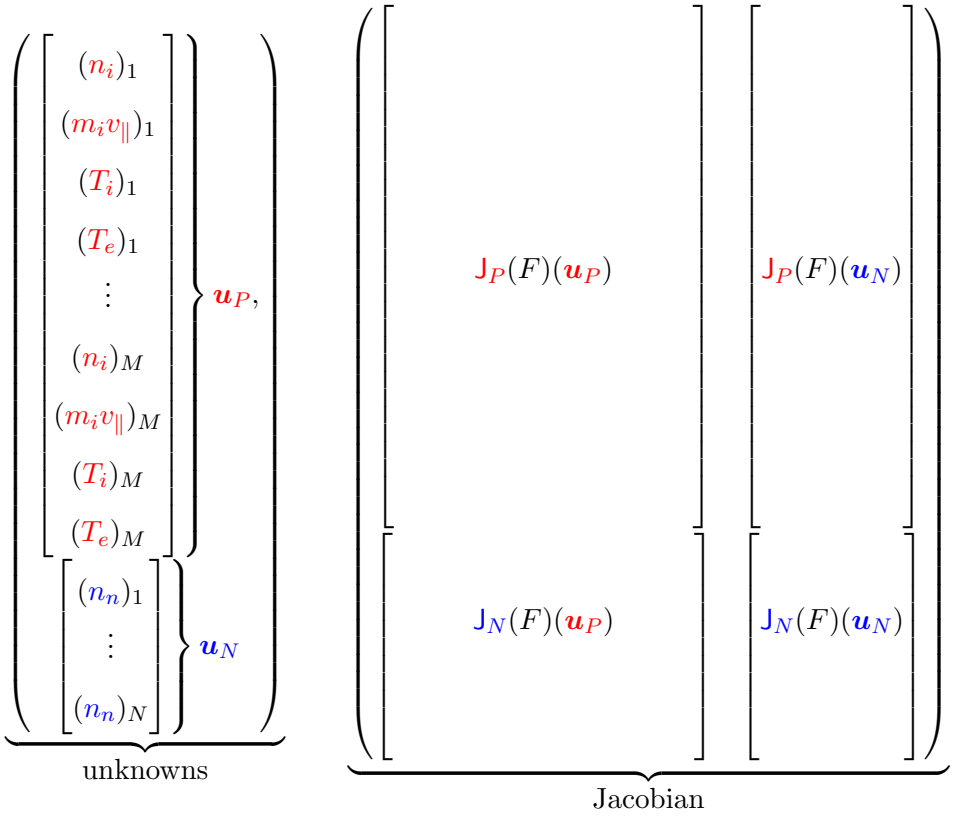
The ASM was shown to be successful when preconditioning the solve involving only the plasma terms in Section 2.3.1. This could be interpreted as part of the system being amenable to a weaker preconditioner which proves ineffective when applied to the coupled system. Along this line of thought, consider the coupled system as organized in the traditional block structure of one block of unknowns

per grid point:

$$\underbrace{\begin{pmatrix} \left[\begin{smallmatrix} n_i \\ m_i v_\parallel \\ T_i \\ T_e \\ n_n \end{smallmatrix}\right]_1 \Bigg\} \boldsymbol{u}_1 \\ \vdots \\ \left[\begin{smallmatrix} n_i \\ m_i v_\parallel \\ T_i \\ T_e \\ n_n \end{smallmatrix}\right]_M \Bigg\} \boldsymbol{u}_M \end{pmatrix}}_{\text{unknowns}}, \quad \underbrace{\begin{pmatrix} \left[\begin{smallmatrix} \\ \mathsf{J}_1(F)(\boldsymbol{u}_1) \\ \\ \\ \\ \end{smallmatrix}\right] & \cdots & \left[\begin{smallmatrix} \\ \mathsf{J}_1(F)(\boldsymbol{u}_M) \\ \\ \\ \\ \end{smallmatrix}\right] \\ \vdots & \ddots & \vdots \\ \left[\begin{smallmatrix} \\ \mathsf{J}_M(F)(\boldsymbol{u}_1) \\ \\ \\ \\ \end{smallmatrix}\right] & \cdots & \left[\begin{smallmatrix} \\ \mathsf{J}_M(F)(\boldsymbol{u}_M) \\ \\ \\ \\ \end{smallmatrix}\right] \end{pmatrix}}_{\text{Jacobian}}$$

where $\mathsf{J}_k(F)(\boldsymbol{u}_\ell)$ is the Jacobian of F with respect to the set of unknowns $\boldsymbol{u}_k$ evaluated at the set of unknowns $\boldsymbol{u}_\ell$, assuming that there are M total unknowns. The matrix above is a $M \times M$ block matrix, with 5×5 blocks. If the vector of unknowns were reordered so that the neutral terms were all segregated from the

plasma terms, the resulting ordering would produce the Jacobian



where now $\mathsf{J}_P(F)(\boldsymbol{u}_N)$ is the Jacobian of F with respect to the plasma terms evaluated at the neutral variables $\boldsymbol{u}_N$, assuming again that there are M total unknowns. This reordered matrix is a 2×2 block matrix with blocks of varying size: $4M \times 4M$ for $\mathsf{J}_P(F)(\boldsymbol{u}_P)$, $4M \times M$ for $\mathsf{J}_P(F)(\boldsymbol{u}_N)$, $M \times 4M$ for $\mathsf{J}_N(F)(\boldsymbol{u}_P)$, $M \times M$ for $\mathsf{J}_N(F)(\boldsymbol{u}_N)$.

One might ask about the purpose of this reordering, as doing so will likely impact the sparsity structure of the matrix which was already well banded in the 1D partition case as shown in Figure 2.5a. While true for the full LU factorization, if the nonzeros moved away from the diagonal were neglected during the preconditioning, then the nonzero structure would not be adversely affected. Of course this would be detrimental to the quality of such a preconditioner, but if the terms being neglected were already unimportant, then the terms that were retained would be

sufficient for the linear solve.

Applying this logic to the reordered Jacobian from this problem would produce

$$\mathsf{J}(F)(\boldsymbol{u}) = \begin{pmatrix} \mathsf{J}_P(F)(\boldsymbol{u}_P) & \mathsf{J}_P(F)(\boldsymbol{u}_N) \\ \mathsf{J}_N(F)(\boldsymbol{u}_P) & \mathsf{J}_N(F)(\boldsymbol{u}_N) \end{pmatrix} \longrightarrow \mathsf{P}_{BJ} = \begin{pmatrix} \mathsf{J}_P(F)(\boldsymbol{u}_P) & \\ & \mathsf{J}_N(F)(\boldsymbol{u}_N) \end{pmatrix} \quad (2.3)$$

where P_{BJ} is a block Jacobi preconditioner void of the off-diagonal blocks present in the true Jacobian. There are two key benefits to considering this preconditioner over a strictly algebraic preconditioner such as SSOR or ILU(k) or the domain decomposition preconditioner ASM:

1. By neglecting the off-diagonal coupling terms $\mathsf{J}_P(F)(\boldsymbol{u}_N)$ and $\mathsf{J}_N(F)(\boldsymbol{u}_P)$ the system is broken into smaller sub problems, each of which can potentially be handled by a different solver.
2. The plasma and neutral terms have now been segregated. As was described in Section 2.3 these two systems have different physics and solving the systems together was very ill-conditioned. Now the most ill-conditioned terms can be handled separately from other variables.

Combining these two ideas with the preconditioning results from Section 2.3.1 and Section 2.3.3 provides a logical method of inverting the two diagonal blocks from P_{BJ}: $\mathsf{J}_P(F)(\boldsymbol{u}_P)^{-1}$ will be produced with ASM and $\mathsf{J}_N(F)(\boldsymbol{u}_N)^{-1}$ will be computed with a stronger preconditioner. In this way each system requires exactly the amount of care demanded by the component physics.

The classical method of matrix splitting would expect the system to be written

as $\mathsf{J}(F)(\boldsymbol{x}) = \mathsf{A} - \mathsf{B}$ producing a linear iteration of the form

$$\begin{aligned}
\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{\delta} &= -F(\boldsymbol{u}) \\
(\mathsf{A} - \mathsf{B})\boldsymbol{\delta} &= -F(\boldsymbol{u}) \\
\mathsf{A}\boldsymbol{\delta}_k &= -F(\boldsymbol{u}) + \mathsf{B}\boldsymbol{\delta}_{k-1} \\
\boldsymbol{\delta}_k &= -\mathsf{A}^{-1}F(\boldsymbol{u}) + \mathsf{A}^{-1}\mathsf{B}\boldsymbol{\delta}_{k-1}
\end{aligned}$$

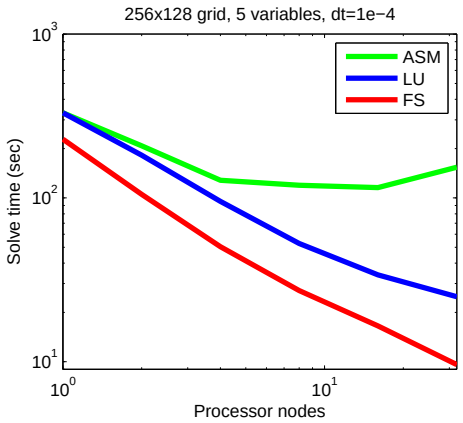
whose convergence depends upon the spectrum of $\mathsf{A}^{-1}\mathsf{B}$. Here, since the problem is being solved in the Krylov subspace framework this exact iteration need not take place because convergence is guaranteed by GMRES. Instead, the splitting is only a preconditioner, and as such it need not produce the exact solution. This allows us to treat the P_{BJ} matrix as A in the splitting; the off-diagonal blocks are B which is actually disregarded during the preconditioning. In PETSc [8], this splitting is termed *Additive FieldSplit* (FS) because the individual blocks are inverted separately and then the results are added together.

2.5 Numerical results for field-split

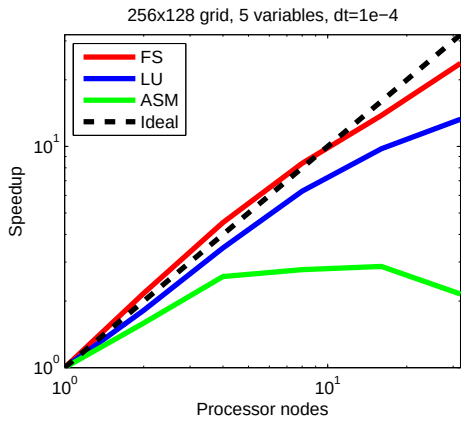
In this section we will experiment with the field-split preconditioner on several simulations. The first experiment reproduces the five variable simulation from Section 2.3.5 and demonstrates that by splitting the preconditioning, we can significantly improve the scalability of the solver. We then increase the time step (and increase the condition of the system as described in Section 2.3.2) and study the effect on the new preconditioner. Section 2.5.3 studies the inclusion of the parallel neutral velocity in the simulation, and the final set of experiments includes a Neon impurity which increases the number of unknowns per grid point to sixteen. In each of these instances, the field-split preconditioner outperforms existing methods.

2.5.1 Comparison to earlier results

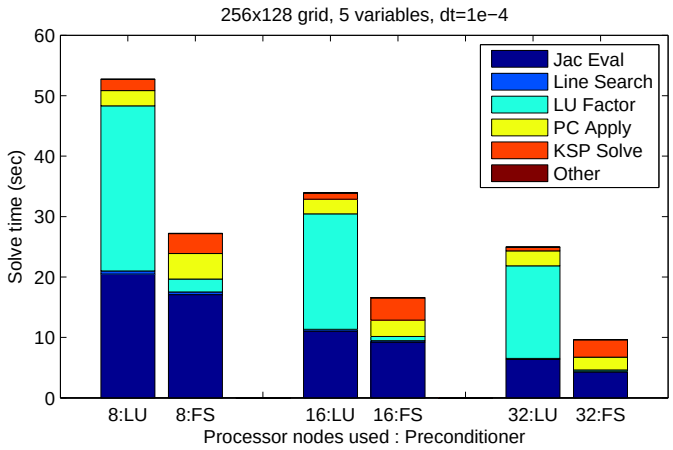
We implement the approach proposed in the Section 4.4 using PETSc's latest developed *FieldSplit* (FS) preconditioner, which allows users to describe the individual blocks of the Jacobian matrix and then "tell" the solver how to compose the full preconditioner based on smaller inner preconditioners associated with blocks and Schur complements. With this new approach to preconditioning - handling the plasma and neutral terms separately - there is improvement in the solve speed and scalability, as shown in Figure 2.10.



(a) FS solves are much faster than LU



(b) FS scales better than LU and ASM



(c) Little increase in LU factorization cost

Figure 2.10: FieldSplit preconditioning outperforms LU at $\Delta t = 10^{-4}$

NP	Avg. linear iterations			Solve time (sec)		
	LU	ASM	FS	LU	ASM	FS
1	7	7	13	331	331	228
2	7	87	17	176	208	105
4	7	136	17	95	128	50
8	7	255	22	54	119	27
16	7	650	32	35	115	17
32	7	1585	44	26	154	10

Table 2.6: FieldSplit iterations are fewer and less costly.

Gains made by the FieldSplit preconditioner over the global LU come from the reduced cost of the LU factorization within FieldSplit. Recall that the two disjoint blocks present in the preconditioner are inverted with two different methods:

- $\mathsf{J}_P(F)(\boldsymbol{u}_P)^{-1}$ is computed via ASM where the domain is divided into n partitions, where n is the number of processors present. Each domain, along with a small overlap from neighboring domains, is then solved with LU factorization. Each LU factorization is then on a block roughly $1/n$ times the size of $\mathsf{J}_P(F)(\boldsymbol{u}_P)^{-1}$ and requires no communication for each component factorization.
- $\mathsf{J}_N(F)(\boldsymbol{u}_N)^{-1}$ is computed via Algebraic Multigrid (using HYPRE) over the entire domain without any terms lost because of the parallel partitioning. This requires communication between processes during the factorization and becomes less efficient as the number of processors increases because less work is being done by each core without a reduction in the cost of communication.

Other solves can be conducted on these blocks, including a Block Jacobi solve on the plasma block allowing for no overlap between domains and full LU on the neutral block which would be appropriate given its ill-conditioning. Initially though, the approach described above is sufficient to demonstrate the potential of FieldSplit

on the edge plasma transport problem. Figure 2.10a shows the FieldSplit preconditioner consistently outperforming the LU preconditioner to the point where for $NP = 32$ the FieldSplit is roughly 2.5 times faster.

For the LU preconditioner, the cost of computing the decomposition is scaling very poorly, as should be expected because of the level of fill-in which requires additional memory allocation and inter-processor communication. It is also apparent that the **PC Apply** is failing to scale as the size of the yellow bar is constant. Given the fact that the number of linear iterations per nonlinear iteration is constant at 7, this extra cost is likely caused by the increase in communication required to perform the triangular solve: as NP increases, the share of the relevant vectors and matrices stored on each processor decreases requiring extra data transfer given the same nonzero structure.

Within FS, the cost of **LU Factor** drops to negligible, mostly because the size of the subdomains conducting those factorizations gets shrinks. The increase in linear iterations per nonlinear iteration as seen in Table 2.6 causes the **PC Apply** and **KSP Solve** bars to maintain their size rather than continuing to scale as well as they could. This corresponds then to a small loss in scalability for $NP = 32$ (as seen in Figure 2.10b) because the **LU Factor** bar is already as small as possible and the other linear solve terms are bounded by the number of linear iterations.

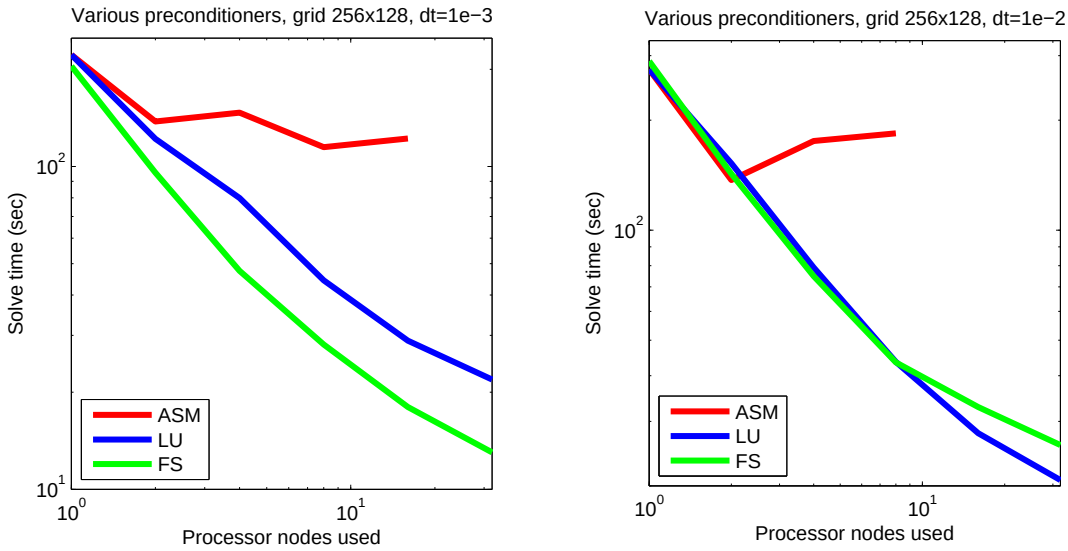
To consider things more qualitatively, FS can be thought of as a targeted preconditioner which uses slower techniques where necessary. Its performance is predicated on being able to solve the plasma terms effectively with ASM even though Figure 2.10a shows that ASM scales very poorly on the plasma/neutral coupled system. FieldSplit takes advantage of the results from Figure 2.6b which show that ASM works well on the plasma terms, and uses a slower, more powerful solver

(Algebraic Multigrid) on the tougher terms which prevent ASM from working well.

In conclusion, the significant improvement from LU preconditioning to Field-Split can be attributed to selective use of a domain decomposition preconditioner on variables which do not demand global coupling during the solve.

2.5.2 Larger time steps

Many significant tokamak dynamics can be observed with small time steps (refer to Figure 2.1), but in order to conduct a steady-state simulation, larger time steps may be appropriate. Unfortunately, as discussed in Section 2.3.2, some preconditioners fail to perform when applied to larger time steps. Figure 2.11 shows results for time steps 10 and 100 times larger than previous experiments.



(a) FieldSplit outperforms LU for $\Delta t = 10^{-3}$

(b) FieldSplit falters near $\Delta t = 10^{-2}$

Figure 2.11: FieldSplit performs adequately for $\Delta t < 10^{-2}$.

The LU preconditioner performs independently of Δt because the time step does not change the nonzero structure of the matrix. The same cannot be said

NP	Preconditioner		
	LU	FS	ASM
1	7	25	7
2	7	26	52
4	8	22	103
8	8	33	240
16	8	43	621
32	8	53	**

(a) Linear iterations per nonlinear iteration for $\Delta t = 10^{-3}$

NP	Preconditioner		
	LU	FS	ASM
1	7	39	7
2	7	46	51
4	6	44	180
8	6	56	427
16	6	89	**
32	6	113	**

(b) Linear iterations per nonlinear iteration for $\Delta t = 10^{-2}$

Table 2.7: There is a distinct increase in the needed number of linear iterations between $\Delta t = 10^{-3}$ and $\Delta t = 10^{-2}$.

for the FieldSplit preconditioner, whose quality decreases with an increase in time step. This is in agreement with Section 2.3.2 and is likely brought on by the increasingly poor performance of ASM. In both Figure 2.11a and Figure 2.11b the Additive Schwarz preconditioner ASM fails to converge for larger NP and thus FieldSplit suffers accordingly.

These results bring to the forefront the significance of the ASM component of the FieldSplit preconditioner. FieldSplit outperforms the LU preconditioner when large portions of the system do not require the full LU factorization, and thus are well-conditioned. As the time step increases, the plasma terms become more difficult to solve and the ASM preconditioner fails to provide an adequate speedup over the LU preconditioner. This may motivate a more general process for identifying appropriate FieldSplit components: sets of variables which can be solved cheaply should be isolated from those that require more attention. For this particular problem the fields were chosen for physics reasons, but here it is obvious that, despite the physics, the plasma terms cease to be sufficiently trivial given larger time steps.

It should also be noted that the quality of the initial guess will generally degrade

given a larger time step. For this example there is no significant effect, but it is conceivable that the choice of initial guess could have an effect on the convergence of FieldSplit given the longer path that each linear solve takes to convergence.

2.5.3 Solving for the neutral parallel velocity

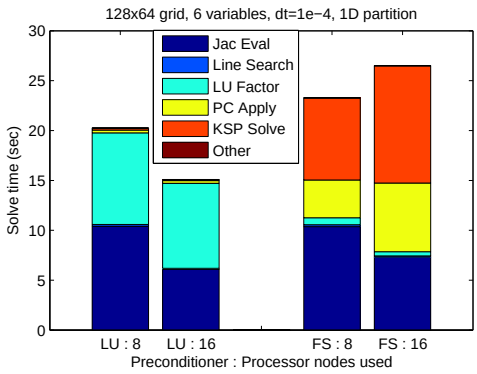
One of the main goals of studying different preconditioning techniques is to allow for flexibility in the solver for different physical simulations. In addition to the 5 degrees of freedom which exist in experiments thus far (D^+ temperature, D^+ velocity, D^+ density, e density, D density), it is also possible to solve for the D parallel velocity in the system of differential equations; until now it has been computed algebraically using the solution to the 5 variable differential system as described in Section 2.2. Doing so increases the share of equations describing the neutral terms from 20% to 33% (from 1 of 5 to 2 of 6).

It is reasonable to assume that this shift will have an effect on the solve strategy - for 5 variables the 1D domain decomposition was preferred by the neutrals and thus was preferred by FieldSplit because only a single neutral term dissented. Doubling the number of neutral terms has shifted the balance of power, as can be seen in Figure 2.12 where the 2D discretization is preferred to the 1D discretization.

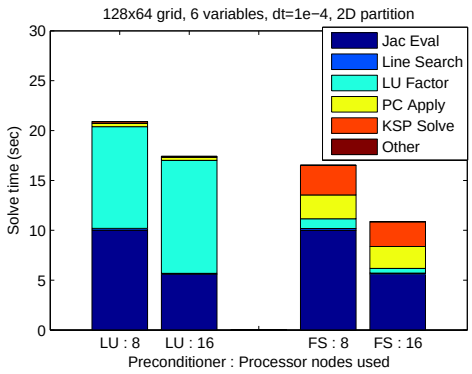
NP	**Preconditioner**		
	LU	FS	
		1D	2D
8	2	89	41
16	2	175	53

Table 2.8: There is a clear difference in average linear iterations per nonlinear iteration

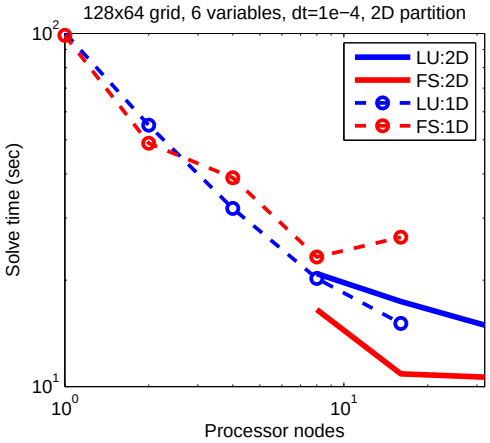
One small thing to note is that the NP=4 2D case does not exist, because of



(a) In 1D FieldSplit fails to perform



(b) In 2D FieldSplit outperforms LU preconditioner



(c) 1D and 2D results differ significantly

Figure 2.12: The 6 variable case sees better performance with FieldSplit using a 2D partitioning

restrictions made to the structure of the 2D partitioning - to prevent a cell from having neighboring values both adjacent and across the branch cut, a 4 processor case is not viable. Also of note is that 3 fields are chosen in this FieldSplit preconditioner instead of 2 previously: plasma terms, neutral density, neutral momentum. This means that in the preconditioner, the terms coupling the neutral density with other variables are ignored, as well as the terms coupling the neutral

momentum to other variables. In matrix form this looks like

$$\begin{pmatrix} \mathsf{J}_P(F)(\boldsymbol{u}_P) & \cancel{\mathsf{J}_P(F)(\boldsymbol{u}_N)} & \cancel{\mathsf{J}_P(F)(\boldsymbol{u}_M)} \\ \cancel{\mathsf{J}_N(F)(\boldsymbol{u}_P)} & \mathsf{J}_N(F)(\boldsymbol{u}_N) & \cancel{\mathsf{J}_N(F)(\boldsymbol{u}_M)} \\ \cancel{\mathsf{J}_M(F)(\boldsymbol{u}_P)} & \cancel{\mathsf{J}_M(F)(\boldsymbol{u}_N)} & \mathsf{J}_M(F)(\boldsymbol{u}_M) \end{pmatrix}$$

where $\boldsymbol{u}_M$ are the momentum variables and $\mathsf{J}_M(F)$ is the Jacobian of F with respect to $\boldsymbol{u}_M$.

Obviously, the most significant point in Figure 2.12c is the better performance of FieldSplit on the 2D partition than the 1D partition. This is in direct contrast to the performance in Section 2.5.1 which seems to indicate that a greater contribution by neutral terms requires a shift to a 2D decomposition to maintain solver speed.

The main reason for the improved behavior with the 2D partition is the fewer linear iterations required per nonlinear iteration, as seen in Table 2.8. Because the $\mathsf{J}_N(F)(\boldsymbol{u}_N)$ and $\mathsf{J}_M(F)(\boldsymbol{u}_M)$ blocks are solved directly with the full factorization, the only difference between the 1D and 2D cases is the $\mathsf{J}_P(F)(\boldsymbol{u}_P)$ solve which is conducted with ASM using sequential LU on each subset of plasma terms. By ordering the plasma variables into 2D blocks rather than 1D strips, more significant terms are retained making the preconditioner more effective. It is interesting that for this experiment the plasma terms prefer the 2D decomposition whereas when treated in Section 2.3.1 separately they prefer the 1D decomposition.

The LU preconditioner stumbled when applied to the 2D partition because more terms appeared off diagonal causing more fill-in during the factorization and communication during the triangular solves; recall the difference in structure shown in Figure 2.5. Unlike the FieldSplit preconditioner, there was no possible gain in reducing the number of linear solve iterations because the LU preconditioner was already taking as few as possible.

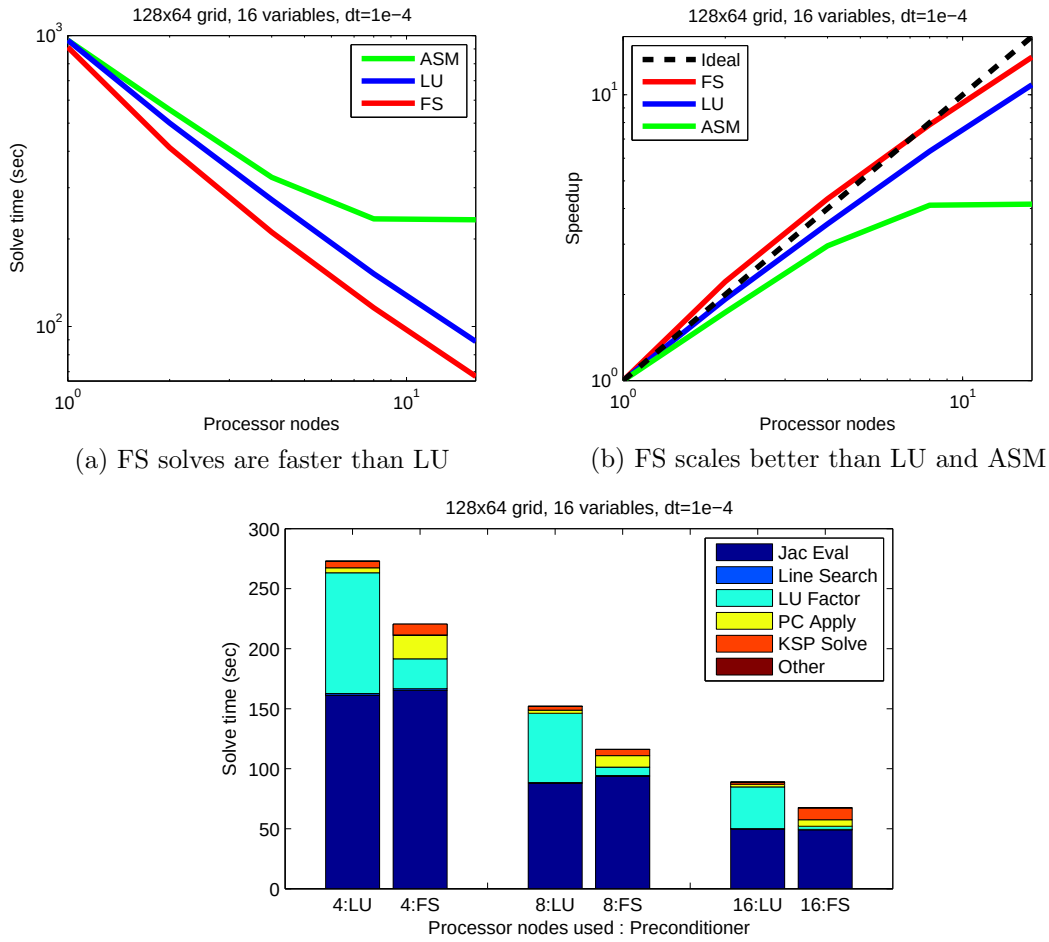
2.5.4 Adding a Neon impurity

It has been established that impurity seeding of a plasma discharge can decrease turbulence and improve confinement [125]. Introducing a Neon impurity to the UEDGE simulation increases the complexity of the simulation because 10 new ion densities must be computed (one for each possible charge that a Neon could take) as well as a neutral Neon species. For a UEDGE simulation, this increases the total degrees of freedom at each grid point in the simulation to 16, with 5 original deuterium variables and 11 Neon variables - the neutral deuterium and Neon momentums with be solved for algebraically and not included in the system of differential equations.

Using a 128x64 grid, experiments were conducted using LU, ASM and FieldSplit preconditioners; the outcomes are compared in Figure 2.13. The FieldSplit performed here resembles the structure from Section 2.4 where all the plasma terms (now including the Neon ions) are segregated from the neutral terms (now including neutral Neon). Table 2.9 shows that both FieldSplit and LU incur almost no change in average linear iterations - this coupled with the diminishing cost of FieldSplit for more processors explain the better scalability on display in Figure 2.13b. It should be noted that on this smaller grid it is only possible to run up to $NP = 16$ for the 1D partition.

NP	**Average KSP its**			**Solve time (sec)**		
	LU	ASM	FS	LU	ASM	FS
1	8	8	21	964	964	913
2	8	79	19	500	556	411
4	8	128	21	273	326	211
8	8	281	29	152	235	116
16	8	622	51	89	233	67

Table 2.9: FieldSplit iterations are fewer and less costly for Neon simulations.



(a) FS solves are faster than LU

(b) FS scales better than LU and ASM

(c) Different performance for FS and LU

Figure 2.13: FieldSplit preconditioning outperforms LU when simulating a Neon impurity

Although this simulation involves more unknowns than the 5 variable (Section 2.5.1) and 6 variable (Section 2.5.3) cases, it is being run on the same size mesh. The greater degrees of freedom per node create a larger system with similar block structure, but larger blocks which are more dense. This is in contrast to a system with 5 active variables being run on a 256x128 mesh which would have roughly the same size, but a very different, and more sparse, nonzero pattern. Understanding the structure of the system is crucial to choosing an appropriate preconditioner, and this is especially true for FieldSplit given the free parameters needed during its construction (selecting fields, choosing coupling, sub solvers/parameters).

2.6 Summary

Jacobian-Free Newton Krylov is a powerful tool for solving nonlinear systems, but it requires the use of preconditioning, which may be costly. Traditional preconditioners have been algebraic in nature, requiring only the operator in question and manipulating it to some easily invertible approximation of the true system. Physics-based preconditioning is the idea that leveraging knowledge of the physics which created the operator can produce a better preconditioner absent any purely algebraic motivation.

For the system discussed here, our numerical experiments demonstrate that two different physical phenomena exist in the coupled plasma/neutral simulation. Because existing preconditioners either fail to perform or scale poorly, we have developed a new preconditioner. By separating the preconditioner into neutral and plasma components, each is solved with an appropriate method to maximize scalability. This approach is called FieldSplit within the PETSc library, and elsewhere may be referred to as operator-specific preconditioning.

By preconditioning the troublesome neutral terms directly and the plasma terms with a domain decomposition solver in Section 2.4 the maximum scalability is preserved while still solving the system with both sets of variables. Figure 2.10 shows this operator-specific preconditioner to be faster and have greater scalability than using the LU or ASM preconditioner on the full system. This technique of breaking a coupled problem into components and handling the components and the coupling individually can be applied to more complicated problems in magnetic confinement fusion, including the addition of a neutral deuterium momentum term or a Neon species.

The main contributions of this chapter are the insights gained on the computational complexity of the coupled plasma/neutral transport problem, and the associated FieldSplit preconditioner developed to confront these problems. This FieldSplit approach to physics preconditioning will be used in Chapter 6 to solve a different coupled problem. This chapter has shown that the flexibility of FieldSplit gives it the potential to be more efficient than the LU preconditioner when applied to stiff problems for which a full factorization was traditionally required. It is however incumbent upon the user to convert insights about the physical system into the structure of the FieldSplit preconditioner to create the most efficient solver.

FieldSplit can be augmented to fit the specifics of an alternate problem beyond those discussed here. Some of these ideas can be implemented quickly in the PETSc framework, while others require more programming. Of course, the validity of any of these methods is subject to the constraints of the problem under consideration, and thus evidence (either analytical or computational) should be gathered to support their use.

If the off-diagonal coupling terms of the Jacobian were needed to speed up the convergence of the iterative solver, more complicated matrix splittings are available. Each of these have the ability to retain the coupling terms present in the reordered Jacobian, as well as potentially using the Schur complement which is currently available in PETSc. The Schur Complement [84] of a matrix

$$\begin{pmatrix} \mathsf{A}_1 & \mathsf{A}_2 \\ \mathsf{A}_3 & \mathsf{A}_4 \end{pmatrix}^{-1} = \begin{pmatrix} \mathsf{J}_P(F)(\boldsymbol{u}_P) & \mathsf{J}_P(F)(\boldsymbol{u}_N) \\ \mathsf{J}_N(F)(\boldsymbol{u}_P) & \mathsf{J}_N(F)(\boldsymbol{u}_N) \end{pmatrix}^{-1}$$

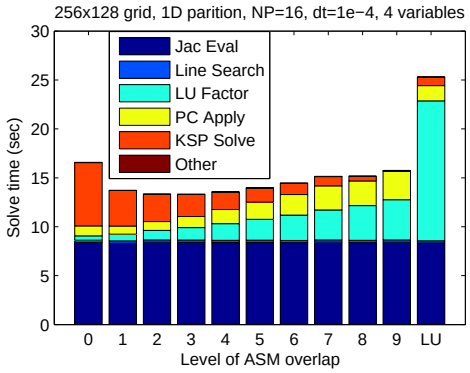
is written as

$$\underbrace{\begin{pmatrix} \mathsf{I} & 0 \\ -\mathsf{A}_4^{-1}\mathsf{A}_3 & \mathsf{I} \end{pmatrix}}_{C_L} \underbrace{\begin{pmatrix} (\mathsf{A}_1 - \mathsf{A}_2\mathsf{A}_4^{-1}\mathsf{A}_3)^{-1} & 0 \\ 0 & \mathsf{A}_4^{-1} \end{pmatrix}}_{C} \underbrace{\begin{pmatrix} \mathsf{I} & -\mathsf{A}_2\mathsf{A}_4^{-1} \\ 0 & \mathsf{I} \end{pmatrix}}_{C_R}$$

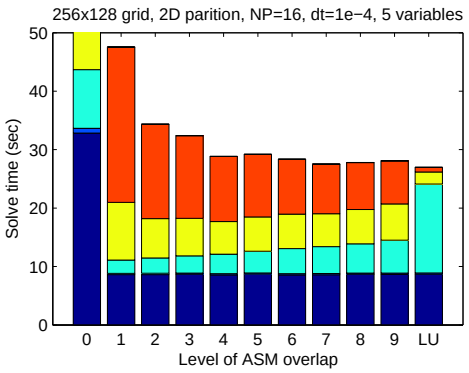
and this multiplicative decomposition which would maintain the block diagonal spirit of P_{BJ} without fully disregarding the off-diagonal terms. When the off-diagonal block are needed, this provides a mechanism to include them, albeit at additional cost.

Beyond changing the preconditioner to improve its quality, there may also be improvements on how the blocks $\mathsf{J}_P(F)(\boldsymbol{u}_P)^{-1}$ and $\mathsf{J}_N(F)(\boldsymbol{u}_N)^{-1}$ within Additive FieldSplit are computed so as to improve speed. One possible goal would be to determine what level of overlap is appropriate for the plasma terms. As Figure 2.14 shows when ASM is applied to the full system, increasing the retained overlap between domains improves the speed of the solve.

When the plasma terms are considered separately in Figure 2.14a, as they are in FieldSplit, this improvement seems limited as the reduction in the number of linear iterations appears counterbalanced by the increase in **LU Factor** for sufficiently large overlap. For both plasma and neutral terms acting together on a 2D partition as seen in Figure 2.14b, the best performance is for very high overlap which confirms our belief that adding neutral terms to the problem requires a much stronger solver.



(a) There is an optimal ASM overlap for plasma terms in 1D.



(b) When the neutral terms are considered, the optimal overlap is near the full LU.

Figure 2.14: The best ASM overlap is dependent on many factors, including: the problem type, the domain partition and the number of processors.

CHAPTER 3

FINITE DIFFERENCE ACCURACY FOR NONLINEAR SYSTEMS

3.1 Introduction

In Section 1.2 we introduced Jacobian-free Newton-Krylov as a method for solving nonlinear systems $F(\boldsymbol{u}) = 0$. At each Newton iteration, the system

$$\mathsf{J}(F)(\boldsymbol{u}^n)\Delta\boldsymbol{u}^{n+1} = -F(\boldsymbol{u}^n) \tag{3.1}$$

must be solved, where $\boldsymbol{u}^n$ is the nth Newton iterate, $\Delta\boldsymbol{u}^{n+1}$ is the next Newton step to be taken, and $\mathsf{J}(F)(\boldsymbol{u}^n)$ is the Jacobian of F evaluated at $\boldsymbol{u}^n$. The initial guess $\boldsymbol{u}^0$ needs to be provided, and often the full search direction $\Delta\boldsymbol{u}^{n+1}$ is reduced by a factor $0 < \alpha < 1$ to help accommodate poor initial guesses. These points are not discussed in this chapter so that we can focus on the specific goal of solving the linear system.

We are generally interested in solving (3.1) iteratively, because many physical discretizations of continuous problems yield sparse or well-structured matrices. Because actual computation of $\mathsf{J}(F)(\boldsymbol{u}^n)$ is impractical we often instead use approximate Jacobian-vector products to for the requisite Krylov space. The values $\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b}$ can be approximated via finite differences:

$$\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b} \approx \frac{F(\boldsymbol{u} + h\boldsymbol{b}) - F(\boldsymbol{u})}{h}.$$

This idea is discussed in [50, 27], among other references. It has become popular in many applications because it eliminates the need for the true Jacobian, and because of the ease with which it can be applied: all that is needed is the function F and a differencing parameter h. Researchers in biogeochemical transport [90], fission

reactors [86], conjugate heat transfer [183], hydrodynamics [101], fluid-structure interaction [174] and many other fields have successfully used this finite difference approximation in their work. We do not intend to debunk the value of this method, but rather propose that it can be improved with a reasonable amount of additional computation.

This chapter begins by introducing some existing methods for choosing a differencing parameter; Section 3.1.1 discusses the logic behind these choices. Section 3.2 describes examples where the existing methods have limited accuracy, and identifies the various scales present in the solution as the cause of this trouble. Our main contribution in this chapter appears in Section 3.3, where an algorithm involving split finite differencing is used to improve the accuracy of the finite difference approximation. This approach is applied in Chapter 6.

3.1.1 Current differencing parameter choices

There are many ways to determine the h parameter which is used to approximate Jacobian-vector products

$$\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b} \approx \frac{F(\boldsymbol{u}+h\boldsymbol{b}) - F(\boldsymbol{u})}{h}.$$

It is assumed that $\boldsymbol{b}$ is a unit vector, since any magnitude could be absorbed in the h term. An appropriate choice should balance two sources of error [107]: the error from the Taylor series approximation which is minimized for a small h, and the error from the cancelation which occurs when h is small and $F(\boldsymbol{u}+h\boldsymbol{b})$ and $F(\boldsymbol{u})$ are close. Thorough analysis of this idea and the following methods for finite differencing was performed in [182].

The most basic choice for balancing the need for a large h (near 1 to minimize

cancelation) and a small h (near unit roundoff to minimize truncation error) would be to split the difference and choose

$$h_b := \sqrt{\epsilon_{\text{mach}}},$$

where ϵ_{mach} is machine precision. In the future, this will be referred to as "Basic". This choice is geometrically half way between ϵ_{mach} where no truncation error exists and 10^0 where no cancelation exists.

Another common choice comes from [135], considers the magnitude of the two vectors involved to produce

$$h_p := \frac{\sqrt{\epsilon_{\text{mach}}}}{\boldsymbol{b}^T\boldsymbol{b}}\sqrt{1 + \|\boldsymbol{u}\|_2}. \tag{3.2}$$

Named after the authors, we will refer to this method as "Walker-Pernice". Although this approach is more costly than h_b, it has the advantage of introducing some consideration of the scale of the problem, which is significant as was discussed earlier. The cost is also reduced significantly when using this approach to compute a full finite difference Jacobian as the vector $\boldsymbol{u}$ is constant for all rows, thus saving many norm computations.

The most popular choice in Jacobian-free Newton-Krylov literature will be named "Dennis-Schnabel", and comes from [50]. This is a more complicated equation designed to incorporate the relationship between $\boldsymbol{u}$ and $\boldsymbol{b}$:

$$h_d := \frac{\sqrt{\epsilon_{\text{mach}}}}{\boldsymbol{b}^T\boldsymbol{b}}\,\sigma_d \tag{3.3}$$

where

$$\sigma_d = \begin{cases} \boldsymbol{u}^T\boldsymbol{b} & \text{if } |\boldsymbol{u}^T\boldsymbol{b}| > u_{min}\|\boldsymbol{b}\|_1 \\ \text{sign}(\boldsymbol{u}^T\boldsymbol{b})u_{min}\|\boldsymbol{b}\|_1 & \text{else} \end{cases}.$$

The u_{min} parameter here is a tunable parameter whose default is set to be 10^{-6} – the purpose of the parameter is to account for the possibility that $\boldsymbol{u}$ and $\boldsymbol{b}$ are nearly orthogonal. This would be problematic because the differencing parameter could get too small and cancelation error would grow substantially.

3.1.2 Finite differences for multiphysics problems

One problem with the finite difference approach for Newton-Krylov, brought up in [50], appears when the values of $F(\boldsymbol{u})$ are on wildly different scales: this is of significance here because multiphysics problems often present themselves on multiple scales. There are specific examples of the error introduced by this, but for now consider the following reasoning: if one subset of variables $F_1(\boldsymbol{u})$ is on the order of 10^3 and the other variables $F_2(\boldsymbol{u})$ are on the order of 10^{-3} then the convergence of the nonlinear solver is dominated by $F_1(\boldsymbol{u})$. This should be the case because the $F_1(\boldsymbol{u})$ terms contribute much more significantly to the residual $F(\boldsymbol{u})$ despite the fact that $F_2(\boldsymbol{u})$ may still be far from the solution *on its scale*.

The common solution to this problem is to apply a scaling to the function evaluation to accommodate this disparity and move all the variables to the same scale. This technique, discussed in [27], is used in many instances (e.g., the UEDGE simulation from Chapter 2 scaled its components) and should work well as long as there is a typical scale associated with each of the variables. Even if that scale changes during the problem, it is still possible to apply a scaling as long as it is known a priori. In the simplest sense, the problem $F(\boldsymbol{u}) = 0$ is rephrased as

$$F(\boldsymbol{u}) = \begin{pmatrix} F_1(\boldsymbol{u}) \\ F_2(\boldsymbol{u}) \end{pmatrix} \Rightarrow \begin{pmatrix} \mathsf{C}_1 F_1(\boldsymbol{u}) \\ \mathsf{C}_2 F_2(\boldsymbol{u}) \end{pmatrix} = \begin{pmatrix} \mathsf{C}_1 & 0 \\ 0 & \mathsf{C}_2 \end{pmatrix} F(u) \tag{3.4}$$

where C_1, C_2 are positive valued diagonal matrices which scale the F_1 and F_2 vectors. These matrices should be constants determined by the physics of the system, and in the situation described above, $\mathsf{C}_1 = 10^{-3}$ and $\mathsf{C}_2 = 10^3$ would be appropriate choices.

While scaling is an option, it seems unlikely that all situations can produce a C matrix a priori. Producing a different scaling at each time step may be costly or impractical if multiple time steps are being taken simultaneously, or if the system varies wildly within the "neighborhood" of the solution. Theoretically, we could make some restriction that the neighborhood be chosen small enough so that there is a restricted amount of variation, but in a practical setting that may not be reasonable. The likelihood of a poor initial guess is high which is why globalization techniques such as line search are often implemented within a modern Newton-Krylov solver; globalization methods were briefly discussed in Section 1.2.

There may also be some situations where the finite difference approximation produces unphysical results. One example involving groundwater transport appeared in [89], and demonstrated that the finite difference approximation produced a negative concentration. This suggests again that the multiphysics setting is more sensitive and troublesome than a single component individually.

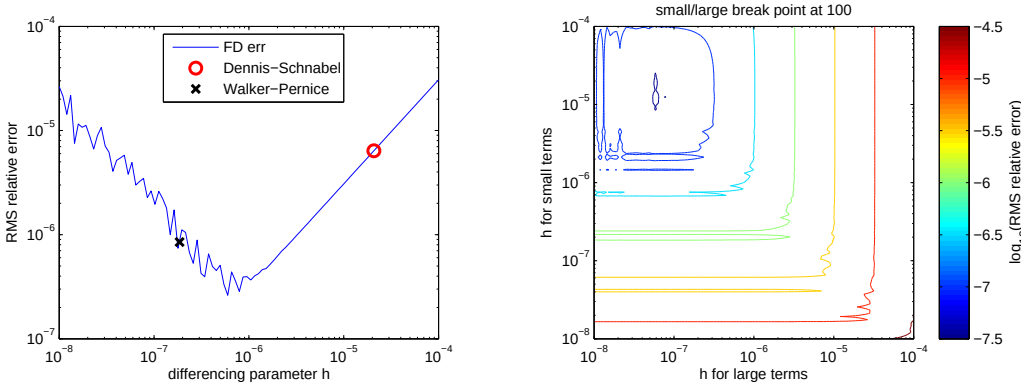
3.2 Shortcomings of finite differencing

As discussed in the previous section, mismatched component scales and an inability to define a small "neighborhood" where the solution exists may cause the full nonlinear system to exist on multiple scales during the Newton iteration. To that end, we may need to consider other methods than the diagonal scaling described

in (3.4) to improve the accuracy of finite difference Jacobian-vector products. We begin with a simple example, where the residual function squares each element of the input vector,

$$F(\boldsymbol{u}) = \boldsymbol{u}^2.$$

Input values are produced by 100 data points evenly distributed in the region $x \in [-10, 10]$, where $(\boldsymbol{u})_i(x) = 1 - x_i^4$, and the vector $\boldsymbol{b}$ is simply a vector of all ones. These $\boldsymbol{u}$ values which are now input for the F evaluation are spread over the range $[-10000, 1]$, and in turn, the output from the $F(\boldsymbol{u})$ evaluation exists over the range $[0, 10^8]$. This wide range of possible outputs impairs the finite difference evaluation, as can be seen in Figure 3.1.



(a) Neither standard guess for h produces optimal accuracy.

(b) If different parameters are used for the largest and smallest terms, the optimal accuracy can be improved.

Figure 3.1: The accuracy of finite difference computation is related to the parameter h but bounded by the complexity of the function.

In Figure 3.1 the term RMS relative error is used; this term is computed by

$$err_{RMS} = \frac{1}{\sqrt{n}} \sqrt{\sum_{k=1}^{n} \left(\frac{FDb_k - Jb_k}{|Jb_k| + \sqrt{\epsilon_{\text{mach}}}} \right)^2}$$

where Jb_k is the k^{th} element of the true Jacobian-vector product, FDb_k is the k^{th} element of the finite difference approximation and n is the number of elements in

the residual evaluation. This method of computing the error makes sure that each value is treated on its own scale, so that one very large value does not dominate the accuracy of the finite differencing. The term $\sqrt{\epsilon_{\text{mach}}}$ appears in the denominator to prevent division by a number less than $\sqrt{\epsilon_{\text{mach}}}$, which would cause values near 0 to unacceptably dominate the error measure.

Figure 3.1a shows that the optimal accuracy for a single differencing parameter used on the entire function is better than any of the parameter approaches described in Section 3.1.1. Of course determining that optimal parameter requires knowledge of the true Jacobian so we are not recommending here that the optimal parameter needs to be found. Rather this picture is supposed to show that one does exist and that finding it is not as trivial as the approaches described earlier. See Table 3.1 for a listing of the notable h values and associated errors.

Choice of h	**Label**	h	err_{RMS}
Basic	h_b	1.0e-8	2.6e-5
Dennis-Schnabel	h_d	2.1e-5	6.4e-6
Walker-Pernice	h_p	1.9e-7	8.5e-7
Optimal	$\tilde{h}$	6.0e-7	2.6e-7
Split	h_1 & h_2	5.9e-8 & 1.3e-5	2.5e-8

Table 3.1: The result of various differencing parameters on the finite difference accuracy.

These results show that, in this example, the best accuracy that can be achieved for the optimal differencing parameter $\tilde{h}$ is 2.6e-7. Many factors contribute to this particular value: the machine precision ϵ_{mach}, the residual function F, the RHS vector $\boldsymbol{b}$, and the range in scales of $F(\boldsymbol{u})$. The first three factors are generally out of our control, and when a single h parameter is used, we are likewise unable to dissociate the various scales present in the problem.

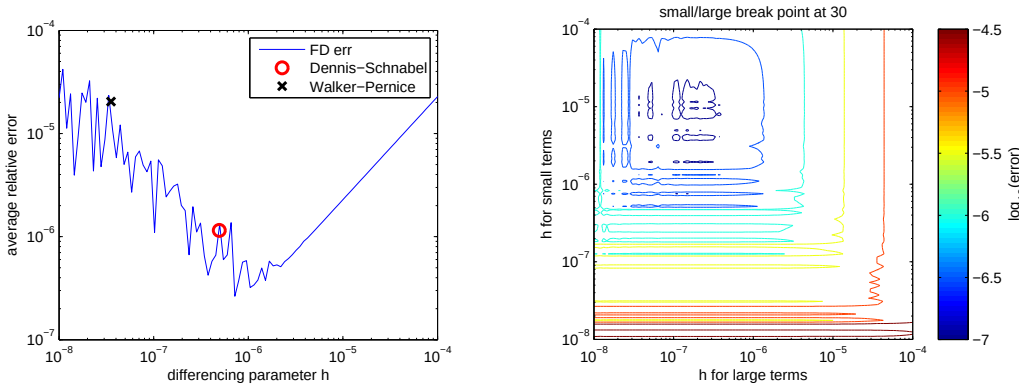
To attempt to separate the scales, we could consider the use of different h

values to compute the Jacobian-vector product for the larger and smaller scales. If we separated our residual into 2 components based on scale, and call h_1 the differencing parameter for the large scale and h_2 the differencing parameter for the small scale, we could choose h_1 and h_2 separately to optimally account for error on only those scales. The full finite difference approximation would be computed once each using h_1 and h_2, but the final approximation would be composed of terms from the large scale using h_1 and from the small scale using h_2.

The results of such an experiment are presented in Figure 3.1b and there is a significant improvement in accuracy to be gained when isolating the largest and smallest components of $F(\boldsymbol{u})$. The break point between scales for this example is chosen to isolate the 32 values of $F(\boldsymbol{u})$ less than 10^4 from the 68 values of $F(\boldsymbol{u})$ greater than 10^4. This choice was arbitrary, and many other break points would have worked as well. When we locate the optimal error on the contour plot, we see it is 2.5e-8, which is a full order of magnitude better than any single h could ever be. All values for the single differencing parameter case are restricted to $h_1 = h_2$ which lies on the anti-diagonal of the contour plot; because the optimal value on that plot does not lie on that line, it would never be possible to be that accurate with only one differencing parameter.

A more realistic example than the simplistic one just noted might involve a 2-pt boundary value problem. Suppose $x \in [-10, 10]$ again, this time with $(\boldsymbol{u})_i(x) = 1-x_i^2$ and the vector $\boldsymbol{b}$ chosen randomly from a Unif(0,1) distribution. The residual function is chosen to be $F(\boldsymbol{u}) = \nabla^2\boldsymbol{u} - \boldsymbol{u}^2$, which is discretized with second order finite differences. The results are on display in Figure 3.2 and Table 3.2.

The improvement that we are seeing here is a result of the large and small terms being handled on their own scales rather than jointly. Drawing a line from



(a) Dennis-Schnabel performs well for this problem.

(b) When the scales are separated, the optimal perfomance is better.

Figure 3.2: For an ODE, there is better accuracy when splitting scales.

Choice of h	**Label**	h	err_{RMS}
Basic	h_b	1.0e-8	2.2e-5
Dennis-Schnabel	h_d	2.1e-5	1.1e-6
Walker-Pernice	h_p	1.9e-7	2.0e-5
Optimal	$\tilde{h}$	6.0e-7	2.6e-7
Split	h_1 & h_2	1.5e-7 & 7.9e-6	5.8e-8

Table 3.2: Various differencing parameters have varying effects on the finite difference accuracy, but no one parameter can match the accuracy of the multiple parameter scheme.

the bottom left to the top right of image Figure 3.1b would show the manifold to which the single h value computation is restricted. As we can see, the optimal values for h lie far off that diagonal meaning that there is a region of accuracy that could never be reached using only a single differencing parameter.

3.3 An algorithm for split finite differences

Now that we have described the potential hazard associated with choosing only a single differencing parameter, we can consider an algorithm which would allow for Jacobian-vector approximation using multiple differencing parameters. We have

already described the basic components in Section 3.2, but we must also define some important terms.

First we define n_t and n_b as the indices associated with the largest and smallest scales respectively. To identify the vector components associated with those indices, we use the notation $\boldsymbol{u}[n_b]$ and $\boldsymbol{b}[n_b]$, and the term $F(\boldsymbol{u}[n_b])$ means to evaluate $F(\boldsymbol{u})$ and then remove all indices not in n_b. The value *tol* is the break point between the largest and smallest scales, such that indices in n_b are chosen to require $|F(\boldsymbol{u}[n_b])| < tol$. The final approximation is denoted by $\boldsymbol{J}_{\boldsymbol{u},\boldsymbol{b}}$.

Algorithm 2 Split Finite Differencing

Given F, $\boldsymbol{u}$, $\boldsymbol{b}$
Choose *tol*
$\boldsymbol{F}_{\boldsymbol{u}} \leftarrow F(\boldsymbol{u})$
Sort $\boldsymbol{F}_{\boldsymbol{u}}$ to find elements less than *tol*
$n_b \leftarrow$ the indices of $\boldsymbol{F}_{\boldsymbol{u}}$ less than *tol*
$n_t \leftarrow$ the indices of $\boldsymbol{F}_{\boldsymbol{u}}$ not present in n_b
Choose h_b based on $F(\boldsymbol{u}[n_b])$ and $\boldsymbol{b}[n_b]$
Choose h_t based on $F(\boldsymbol{u}[n_t])$ and $\boldsymbol{b}[n_t]$
$\boldsymbol{F}_b \leftarrow F(\boldsymbol{u} + h_b\boldsymbol{b})$
$\boldsymbol{F}_t \leftarrow F(\boldsymbol{u} + h_t\boldsymbol{b})$
$\boldsymbol{J}_{\boldsymbol{u},\boldsymbol{b}}[n_b] \leftarrow 1/h_b(\boldsymbol{F}_b[n_b] - \boldsymbol{F}_u[n_b])$
$\boldsymbol{J}_{\boldsymbol{u},\boldsymbol{b}}[n_t] \leftarrow 1/h_t(\boldsymbol{F}_b[n_t] - \boldsymbol{F}_u[n_t])$
return $\boldsymbol{J}_{\boldsymbol{u},\boldsymbol{b}}$

You can notice that this algorithm has 3 "Choose" statements in it, which have been left intentionally ambiguous: one involves the choice of how the splitting between scales is done, and the other two require a choice of differencing parameter. The *tol* parameter is a free parameter which the must input. The other two choices involve how the values h_b and h_t are selected. For future experiments, we choose specific values of h_b and h_t; however, the choice could be automated by using the Walker-Pernice or Dennis-Schnabel method on the components $F(\boldsymbol{u}[n_b])$ and $b[n_b]$ to compute h_b, and a similar approach for h_t.

In the same way that there is more than one way to choose h_b and h_t, there are other ways to split the scales of $F(\boldsymbol{u})$. A fixed number of terms m could be expected for the lower scale, and then the smallest m terms would be included in the index n_b. Of even more practical significance would be some sort of clustering [4] approach to choosing appropriate scale divisions based on the values present in $F(\boldsymbol{u})$. This is an especially promising approach because it takes advantage of information which would not be present a priori and thus could not be used to determine the diagonal scaling which is traditionally applied. Furthermore, this splitting is not fixed across Jacobian-vector products, and thus, unlike the diagonal scaling, a different splitting could be chosen for every product.

3.4 Studying finite differences within a nonlinear solver

Recall that solving nonlinear systems with Newton's method requires solving linear systems of the form (3.1) at each Newton step. In the absence of a true Jacobian, we can only approximate matrix-vector products, which introduces error into the linear system. We use this section to study the effect of such an error on a general linear system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$. Then we consider the potentially negative effect of finite differences on a nonlinear solver for a relatively simple problem. Finally we demonstrate on a more difficult problem the value of the split finite differences algorithm in improving the accuracy of the nonlinear solver.

3.4.1 Error analysis for finite differencing

From [84], error bounds are available for linear systems of the form $(\mathsf{A}+\epsilon\mathsf{F})\boldsymbol{x}(\epsilon) = \boldsymbol{b}+\epsilon\boldsymbol{f}$,

$$\begin{aligned}\frac{\|\boldsymbol{x}(\epsilon)-\boldsymbol{x}\|}{\|\boldsymbol{x}\|} &\leq \epsilon\|\mathsf{A}^{-1}\|\left[\frac{\|\boldsymbol{f}\|}{\|\boldsymbol{x}\|}+\|\mathsf{F}\|\right]+O(\epsilon^2)\\ &\leq \epsilon\kappa(\mathsf{A})\left[\frac{\|\boldsymbol{f}\|}{\|\boldsymbol{b}\|}+\frac{\|\mathsf{F}\|}{\|\mathsf{A}\|}\right]+O(\epsilon^2),\end{aligned} \tag{3.5}$$

where $0 < \epsilon \ll 1$ which makes a Taylor expansion appropriate. When we think about the system we are solving here, we need to consider the amount of error present both in the linear system and the right hand side. The problem here is that the error in the linear system is at a much higher scale because of the finite difference evaluations.

It is reasonable to assume the error present from the finite difference approximation is $O(\sqrt{\epsilon_{\text{mach}}})$ (see [50]), although it may be much greater as we saw in Section 3.2. Note here that we are referring to only the relative error in evaluation of the matrix, not its application to general vectors, although the evaluation can of course be done by matrix vector multiplication with columns of the identity. If we can assume that there is only machine precision relative error in the function evaluation, which is the minimum possible error, then we should have $\|\epsilon\mathsf{F}\| = \sqrt{\epsilon_{\text{mach}}}\|\mathsf{A}\|$ and $\|\epsilon\boldsymbol{f}\| = \epsilon_{\text{mach}}\|\boldsymbol{b}\|$. Plugging this in to (3.5) gives us

$$\begin{aligned}\frac{\|\boldsymbol{x}(\epsilon)-\boldsymbol{x}\|}{\|\boldsymbol{x}\|} &\leq \epsilon\kappa(\mathsf{A})\left[\frac{\epsilon_{\text{mach}}}{\epsilon}+\frac{\sqrt{\epsilon_{\text{mach}}}}{\epsilon}\right]+O(\epsilon^2),\\ &\leq \sqrt{\epsilon_{\text{mach}}}\kappa(\mathsf{A})\left[\sqrt{\epsilon_{\text{mach}}}+1\right]+O(\epsilon^2).\end{aligned}$$

This indicates that for a sufficiently poorly conditioned system we could expect the use of finite differences to be problematic.

3.4.2 A basic example

For this problem, we are interested in solving the 2 point nonlinear BVP

$$\frac{d^2}{dx^2}u + e^{-\sigma u} = 0, \qquad \sigma > 0, \qquad u(-10) = u(10) = 1$$

with a finite difference discretization on 100 evenly spaced points. For $\sigma = 0$ this reduces to just a linear problem with a source, for which the solution $u(x) = 51 - \frac{1}{2}x^2$ can be found analytically. As $\sigma \to \infty$, the nonlinear term fades away and the solution approaches a straight line. An initial guess of all ones is chosen for the nonlinear solve. See Figure 3.3 for an example performance of finite difference approximation as compared to the true Jacobian solve.

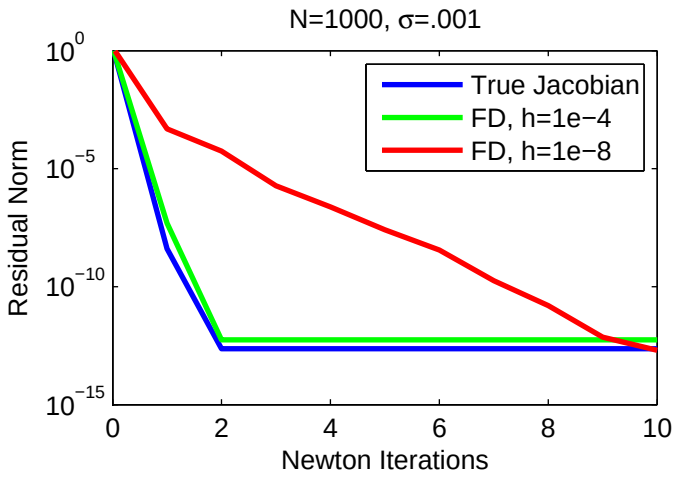


Figure 3.3: Even for a small nonlinearity, the finite difference approximation may stray significantly depending on the choice of differencing parameter.

Although certainly not indicative of all nonlinear solves, this example shows that the presence of error in the linear solve may produce worse Newton iterations. Here each linear solve was performed with preconditioned GMRES, using the true inverse as a preconditioner; this is essentially a direct solve except that the matrix-vector products are not performed with the matrix Jacobian but rather matrix-free methods. The search directions found by the finite difference approximation are a

function of the error in the linear system, which is in turn a function of h.

In Figure 3.3 different h values produce different paths to convergence for the finite difference approximation. This is logical given the previous results but serves to underscore the significance of choosing the "right" h. Even when using more advanced choices of h such as (3.2) or (3.3) issues can arise involving the accuracy of the Δu^k terms being computed. This is especially true at later steps where the magnitude of each step $\|\Delta u^k\|$ is very small. Because the value $\sigma = .001$ is very close to zero, this problem is very nearly linear, and cancelation is less significant than roundoff error. That is why $h = 1e-8$ was much closer to the true Jacobian than $h = 1e-4$.

3.4.3 Improving convergence with split finite differences

Now we use the split finite differences technique to improve convergence for a nonlinear solver. Consider the linear critical gradient model [99], which is sometimes used as a simplified model of nuclear fusion; this was briefly introduced in Section 1.1. The BVP is

$$u_t - (\kappa(u_x)u_x)_x = f, \qquad u(-1) = u(1) = 0,$$

where the diffusivity is a function of the derivative u_x

$$\kappa(u_x) = \frac{\alpha}{2z}\log(\cosh(2zu_x) + \cosh(2zC)) - \alpha C + \frac{\alpha - 2}{2z}\log(2) + \kappa_0 - B, \quad (3.6)$$

and the source is chosen so that the true solution is

$$u(x,t) = e^{-t}(1 - x^2).$$

The parameters appearing in the diffusivity κ determine the nonlinearity in the problem:

- α - The steepness of the nonlinearity
- z - The severity of the change between constant and nonlinear diffusivity
- C - For $|u_x| \ll zC$ the diffusivity is basically constant
- κ_0 - The minimum diffusivity
- B - An integration constant to assure $\kappa(0) = \kappa_0$

Using a finite volume discretization in space and a backward Euler discretization in time produces a nonlinear system at each time step. Attempting to solve for a single time step of size $\Delta t = .1$ with $\Delta x = .04$ (i.e., $N = 50$ total volumes) using both the true Jacobian and finite differences produced Figure 3.4.

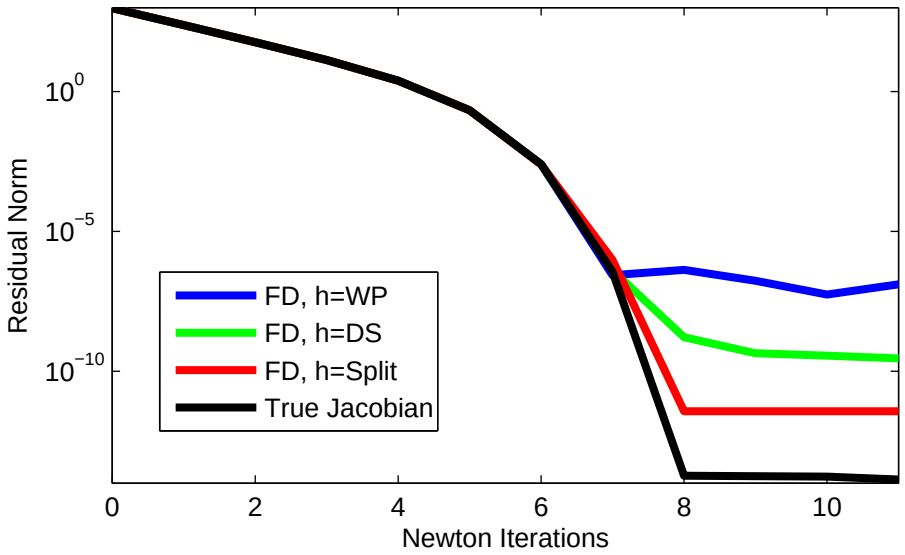


Figure 3.4: Adapting the finite differences using traditional schemes can limit the accuracy of the nonlinear iteration. Greater accuracy can be achieved by using split finite differencing with two differencing parameters. The finite differencing parameters considered are WP=Walker-Pernice (3.2) and DS=Dennis-Schnabel (3.3).

Here the initial guess is chosen as the previous time step (the initial profile) and the diffusivity parameters are all equal to 1. All the finite difference methods

fail to converge as accurately as the true Jacobian, as should be expected given the error in the finite difference approximation. The split finite differencing was much more accurate than either the "Walker-Pernice" (3.2) or the "Dennis-Schnabel" (3.3) methods which are traditionally used.

Here, the splitting strategy used was simple: $F(\boldsymbol{u})$ values were divided in half by magnitude, with larger values differenced with h_t and smaller values differenced with h_b. The differencing values start at $h_t = 10^{-7}$ and $h_b = 10^{-5}$ respectively, and increase gradually up to $h_t = 10^{-5}$ and $h_b = 10^{-3}$ as the residual norm decreases. Recall that for smaller residual vectors $F(\boldsymbol{u})$ the differencing parameter needs to increase, justifying this otherwise ad hoc approach. Going forward, it will be useful to somehow implement the WP or DS methods within the splitting scheme to automate the finite difference choice, as mentioned in Section 3.3.

3.5 Summary

Finite difference Jacobian-vector products are a useful tool for solving nonlinear systems because they allow for linear solves without the full Jacobian. As presented in this chapter, there is limited accuracy attainable for this approach. We propose a new technique for splitting the finite difference approximation into components based on the magnitude of the values in the residual evaluation. This choice helps to isolate the different scales in the function evaluation for situations when the scales are unknown prior to finding the solution, or when the path of the Newton iteration includes values of uncommon scale.

This new technique allows for more accurate finite differences, which in turn may allow for a more accurate nonlinear solve because better search directions

are available. Moreover, as suggested in Figure 3.4, it may not be necessary to always incur the additional cost associated with split finite differencing; only after the residual norm has decreased beyond the accuracy of single finite difference methods is the improvement in accuracy significant. We have not suggested that finite difference methods with only a single differencing parameter are invalid, but rather proposed an additional computational step which may increase the accuracy of approximate matrix-vector products for multiscale systems. Split finite differencing is another tool in our multiphysics infrastructure with the potential to improve the convergence of Jacobian-free Newton-Krylov methods.

There are more advances which can be made on this project. We briefly mention two topics here which are worthy of further thought, but are too complex for this work. The first is a more in depth study of the approach of split finite differencing. Improving accuracy of the Jacobian-vector products with this split approach comes with issues.

Computational cost There is now a need/opportunity to determine this break point between the high and low scales within the function. How is this choice made?

Flexibility Perhaps there are more than 2 scales which should be separated into their own finite difference computations. At what point are you experiencing diminished returns, since each scale requires an additional function evaluation?

The bigger picture Splitting the finite differencing increases the number of function evaluations which need to take place during the linear solve. Why/when would that be worth it when considering the nonlinear solve?

A general algorithm for optimally splitting the finite difference computation must balance the extra time spent computing a more accurate Jacobian-vector product with the time saved during the nonlinear solve by producing a better search direction. Unfortunately, the quality of the path taken during the full Newton solve cannot be determined a priori. One set of tunable parameters which would allow the user to take advantage of the splitting includes

- maximum number of splits - This would cap the total number of function evaluations used to conduct finite differencing.
- suggested splitting points - Given an idea what the scales of the problem are, the algorithm would have a better chance of finding optimal splittings.
- kurtosis - If the user knows the peakedness of the residual function, that would help the algorithm understand the local/global nature of the function, and automatically pick better splits.

The second point of future interest is the study of finite difference approximations as a regularization technique. When Jacobian-vector products are approximated, error is introduced in the system, and the actual problem being solved is

$$(\mathsf{J}(F)(\boldsymbol{u}) + \mathsf{E})\boldsymbol{x} = F(\boldsymbol{u}) + \boldsymbol{\delta}.$$

The matrix E is the error associated with the finite differences, and $\boldsymbol{\delta}$ is the vector of error associated with the function evaluation $F(\boldsymbol{u})$. This $\boldsymbol{\delta}$ vector is typically small (or at least as small as is allowed by the application), but the E matrix may be much larger because there is more error in the finite differencing than the residual evaluation. This error is unwanted because the solution of this system is

further from the optimal search direction, so in pure math we would like to set E to 0 by exactly evaluating the Jacobian.

If, on the other hand, the matrix $\mathsf{J}(F)(\boldsymbol{u})$ were ill-conditioned, then the small error in $\boldsymbol{\delta}$ would potentially be magnified significantly in $\mathsf{J}(F)(\boldsymbol{u})^{-1}\boldsymbol{\delta}$. This could overwhelm the true solution $\mathsf{J}(F)(\boldsymbol{u})^{-1}F(\boldsymbol{u})$, yielding a terrible Newton step. The presence of the error matrix E could actually serve to stabilize the accuracy of search direction by producing a better conditioned system. Research would need to be conducted on the practicality of this method, but this could open up the use of the differencing parameter as a regularization tool. The h, or multiple h values if split differencing is used, would become a free parameter, used to balance accuracy in the Jacobian-vector products with condition of the system. Their value could be determined statistically, as is discussed for a different set of free parameters in Chapter 8.

CHAPTER 4

STABLE INTERPOLATION WITH GAUSSIANS USING EIGENFUNCTIONS

4.1 Introduction

Many multiphysics simulations consist of component models which exist independently and are coupled together at shared boundaries, as discussed in Section 1.3. These component models are often developed without input from each other, meaning that important design choices are made without considering the model to which they will be coupled. This is often the result of two or more successful models being developed in different communities over several years or decades, not the intentional ignorance of an important aspect of the component simulation. Nonetheless, the individuality of the two models has an inimical effect on the simplicity of the coupling between them, because by design neither model is intrinsically aware of the other.

Beyond the complicated physical issues which arise when two disjoint models (with potentially contradictory assumptions about the universe [13, 30, 26, 146]) are coupled, there are also mathematical/computational issues present. One of the issues is the worry that coupling between two high order models will lower the order of accuracy of the full simulation. Another concern is the problem of relating the results of one model as input to the other – because components are often encapsulated, they likely exist on incompatible discretizations. The mechanism by which this data is transferred may be trivial if the grids are chosen very carefully, or mortar methods [58, 7, 108] may be used to produce a special coupling grid between the two models.

Interpolation between the two grids using geometry [98, 19] is also common, although more difficult beyond 1 dimension. The approach of meshfree interpolation to couple two models together is however much less common, mostly appearing only in fluid-structure interaction [3, 177, 178]. Although the meshfree approach to interpolation is a natural choice given its indifference towards the disparate meshes in the components, one of the drawbacks is the potential ill-conditioning associated with kernel methods. This is discussed further in Section 4.2, but if the ill-conditioning were not an issue, meshfree approximation would be an ideal choice for interpolation between models given its freedom to move between meshes and high-order accuracy. Here we address the ill-conditioning issue for the scattered data approximation problem. This chapter lays the groundwork so that Chapter 6 can add meshfree coupling via stable kernel-based interpolation to our multiphysics infrastructure.

Section 4.2 discusses existing literature concerned with the conditioning issue. A possible solution to this conditioning problem is described in Section 4.3, using eigenfunctions of an associated Hilbert-Schmidt operator. A stable basis for performing Gaussian interpolation is derived in Section 4.4, and numerical results are presented in Section 4.5. To reduce the computational cost in higher dimensions, a low-rank expansion of the Gaussians is derived in Section 4.6; numerical results for this technique are presented in Section 4.7. This chapter is based largely on the paper

G. E. Fasshauer, M. McCourt, *Stable evaluation of Gaussian RBF interpolants*, SIAM Journal on Scientific Computing, 2012

and it is cited as [63].

4.2 Moving away from the standard basis

It is well-known that the standard or direct approach to interpolation at locations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \subset \mathbb{R}^d$ with Gaussian kernels

$$K(\boldsymbol{x}, \boldsymbol{z}) = \mathrm{e}^{-\varepsilon^2 \|\boldsymbol{x}-\boldsymbol{z}\|^2}, \qquad \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^d, \tag{4.1}$$

leads to a notoriously ill-conditioned interpolation matrix $\mathsf{K} = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^N$ whenever ε, the so-called *shape parameter* of the Gaussian, is small, i.e., when the set $\{\mathrm{e}^{-\varepsilon^2 \|\boldsymbol{x}-\boldsymbol{x}_j\|^2)}, j = 1, \ldots, N\}$ becomes numerically linearly dependent. This leads to severe numerical instabilities and limits the practical use of Gaussians — even though it is well known that one can approximate a function from the native reproducing kernel Hilbert space associated with the Gaussian kernel with spectral approximation rates (see, e.g., [61, 176]). The fact that most people are content with working in the "wrong" basis therefore has sparked many discussions, including the so-called uncertainty or trade-off principle [60, 151]. This uncertainty principle is tied directly to the use of the standard ("wrong") basis, and we believe it can be circumvented by choosing a better — orthonormal — basis.

The idea of using a "better basis" for RBF interpolation is not a new one. It was successfully employed in [12] to obtain well-conditioned (and therefore numerically stable) interpolation matrices for polyharmonic splines in the context of a domain decomposition method. The technique used there — reverting to a homogeneous modification of the positive definite reproducing kernel associated with the conditionally positive definite polyharmonic spline kernel — was totally different from the one we pursue here. Our basis comes from a series expansion of the positive definite kernel and is rooted in the pioneering work of [127] and [157].

Combining series expansions of the kernel with a QR decomposition of the

interpolation matrix to obtain a so-called RBF-QR algorithm was first proposed in [72] for interpolation with zonal kernels on the unit sphere $\mathbb{S}^2$ and in [70] for interpolation with Gaussian kernels in $\mathbb{R}^2$. These latter two papers motivated the results presented here. The main improvements provided by our work lie in establishing the general connection between the RBF-QR algorithm and Mercer or Hilbert-Schmidt series expansions of a positive definite kernel K defined on $\Omega \times \Omega$ of the form

$$K(\boldsymbol{x}, \boldsymbol{z}) = \sum_{m=1}^{\infty} \lambda_m \varphi_m(\boldsymbol{x}) \varphi_m(\boldsymbol{z}), \qquad \boldsymbol{x}, \boldsymbol{z} \in \Omega,$$

with appropriate positive scalars λ_m and functions φ_m. Here Ω can be a rather general set; however, in this thesis following we focus on $\Omega \subseteq \mathbb{R}^d$ (see Section 4.3.2 for more details).

Having such an expansion allows us to formulate interpolation and approximation algorithms that can be implemented stably in any space dimension. We also consider an alternate highly accurate least-squares approximation algorithm for scattered data fitting with Gaussian kernels that in this form seems to be new to the literature even though general least-squares theory clearly suggests such an approach. In the following section we will discuss the expansion we use for the Gaussian kernel, review the idea of the RBF-QR algorithm and discuss a number of details that are crucial for the implementation of our algorithms. Everything is supported with numerical experiments of Gaussian kernel interpolation and approximation of scattered data in space dimensions ranging from $d = 1$ to $d = 5$.

4.3 An eigenfunction expansion for Gaussians in $L_2(\mathbb{R}, \rho)$

For this section we concentrate on the one-dimensional situation. The generalization to multiple space dimensions d is established in a straightforward manner using the product form of the Gaussian kernel.

It turns out that one can derive (see [137, 184]) a Mercer expansion

$$\mathrm{e}^{-\varepsilon^2(x-z)^2} = \sum_{m=1}^{\infty} \lambda_m \varphi_m(x) \varphi_m(z)$$

for the Gaussian kernel (4.1), with the functions φ_m being orthonormal in $L_2(\mathbb{R}, \rho)$. Here the inner product that determines how we measure orthogonality of functions in $L_2(\mathbb{R})$ is weighted by

$$\rho(x) = \frac{\alpha}{\sqrt{\pi}} \mathrm{e}^{-\alpha^2 x^2}, \qquad \alpha > 0. \tag{4.2}$$

This formulation ensures that the weight function has unit integral, and that the parameter α acts on the same scale as the shape parameter ε of the Gaussian kernel. Moreover, both of these parameters act as length scales for the spatial variable x and use the same units. Since the parameter α determines how the global domain $\mathbb{R}$ is "localized" we can interpret it as a *global scale parameter*.

In order to match up our choice of parameters with those used in [137], we replace the original parameters a, b, and $c = \sqrt{a^2 + 2ab}$ with our own parameters α, β (to be introduced below) and ε in the following way:

$$a = \frac{\alpha^2}{2}, \quad b = \varepsilon^2, \quad c = \frac{\alpha^2 \beta^2}{2}.$$

Using this setup along with the following auxiliary parameters β, γ_m and δ defined

in terms of α and ε, i.e.,

$$\beta = \left(1 + \left(\frac{2\varepsilon}{\alpha}\right)^2\right)^{\frac{1}{4}}, \tag{4.3a}$$

$$\gamma_m = \sqrt{\frac{\beta}{2^{m-1}\Gamma(m)}}, \tag{4.3b}$$

$$\delta^2 = \frac{\alpha^2}{2}\left(\beta^2 - 1\right), \tag{4.3c}$$

the eigenfunctions φ_m of the Gaussian turn out to be

$$\varphi_m(x) = \gamma_m \mathrm{e}^{-\delta^2 x^2} H_{m-1}(\beta\alpha x), \qquad m = 1, 2, \ldots, \tag{4.4a}$$

where H_{m-1} are the *classical Hermite polynomials* of degree $m-1$ defined by their Rodrigues' formula

$$H_{m-1}(x) = (-1)^{m-1}\mathrm{e}^{x^2}\frac{\mathrm{d}^{m-1}}{\mathrm{d}x^{m-1}}\mathrm{e}^{-x^2} \quad \text{for all } x \in \mathbb{R},\ m = 1, 2, \ldots .$$

In order to get a perfect match of our formulas with those in [137], the reader needs to take into account the corrected normalization provided in the errata for [137]. It should be noted that this formulation is related to Mehler's formula and rescaled Hermite functions [165, Problems and Exercises, Item 23]) multiplied by an extra exponential factor due to the localization effects mentioned above.

The corresponding eigenvalues are

$$\lambda_m = \sqrt{\frac{\alpha^2}{\alpha^2 + \delta^2 + \varepsilon^2}}\left(\frac{\varepsilon^2}{\alpha^2 + \delta^2 + \varepsilon^2}\right)^{m-1}, \quad m = 1, 2, \ldots . \tag{4.4b}$$

As already mentioned, the shape parameter ε is related to the *local scale of the problem*, while α is related to the *global scale of the problem*. In addition, the parameter δ also reflects the local scale of the problem. However, while ε gives us the scale of the kernel (which in turn defines the underlying reproducing

kernel Hilbert space along with a length scale reflected in its norm), the auxiliary parameter δ reflects the length scale of the eigenfunctions.

In principle, the parameters α and ε (or α and δ) can be chosen freely. Unfortunately, this choice is not totally independent if one wants a convergent and stable algorithm (see the discussion in Section 4.5.2 for more details). As mentioned in the introduction, the shape parameter ε has important consequences for the stability and accuracy of Gaussian kernel interpolants. In this chapter are generally be interested in small values of ε as this is the range of values of the shape parameter that incurs numerical instability, often with the promise of higher accuracy. It is our goal to circumvent this instability by working with eigenfunctions instead of the usual translates of the Gaussian kernel.

Note that for $\varepsilon \to 0$, i.e., for "flat" Gaussians, and fixed α we always have $\beta \to 1$ and $\delta \to 0$. We see that the eigenfunctions φ_n converge to the normalized Hermite polynomials $\widetilde{H}_{m-1}(x) = \frac{1}{\sqrt{2^{m-1}\Gamma(m)}} H_{m-1}(\alpha x)$, and the eigenvalues behave like $\left(\frac{\varepsilon^2}{\alpha^2}\right)^{m-1}$. This shows that the main source of ill-conditioning of the Gaussian basis is associated with the eigenvalues, and the RBF-QR strategy suggested in [70, 72] can be employed as explained in the next section.

These observations also provide another simple explanation as to why the "flat limit" of a Gaussian interpolant is a polynomial (see, e.g., [21, 52, 74, 112, 113, 115, 153]).

Looking at (4.4b), one can observe that the eigenvalues $\lambda_m \to 0$ exponentially fast as $m \to \infty$ since the inequality $\varepsilon^2 < \alpha^2 + \delta^2 + \varepsilon^2$ is always true. This idea was used in [61, 62] to establish dimension-independent convergence rates for approximation with Gaussian kernels.

4.3.1 Eigenfunction orthonormality

Because the Gaussian kernel is symmetric positive definite, the eigenfunctions are orthonormal in the $L_2(\mathbb{R}, \rho)$ norm. We establish that fact now, primarily using the orthogonality of the Hermite polynomials,

$$\int_{\mathbb{R}} H_{j-1}(x) H_{k-1}(x)\, \mathrm{e}^{-x^2}\, \mathrm{d}x = \sqrt{\pi}\, 2^{j-1} \Gamma(j) \delta_{j,k}, \tag{4.5}$$

where $\delta_{j,k}$ is the Kronecker delta function, as established in [1].

To begin, we insert (4.4b) and (4.2) into the orthogonality equation above to find

$$\begin{aligned}\int_{\mathbb{R}} \varphi_j(x)\varphi_k(x)\rho(x)\mathrm{d}x &= \int_{\mathbb{R}} \gamma_j \mathrm{e}^{-\delta^2 x^2} H_{j-1}(\beta\alpha x) \gamma_k \mathrm{e}^{-\delta^2 x^2} H_{k-1}(\beta\alpha x) \frac{\alpha}{\sqrt{\pi}}\, \mathrm{e}^{-\alpha^2 x^2} \mathrm{d}x\,, \\ &= \gamma_j \gamma_k \frac{\alpha}{\sqrt{\pi}} \int_{\mathbb{R}} H_{j-1}(\beta\alpha x) H_{k-1}(\beta\alpha x)\, \mathrm{e}^{-(2\delta^2+\alpha^2)x^2} \mathrm{d}x\,.\end{aligned}$$

Now we use the substitution $u = \beta\alpha x$, which implies $x = \frac{1}{\beta\alpha} u$, to convert the integral to

$$\int_{\mathbb{R}} \varphi_j(x)\varphi_k(x)\rho(x)\mathrm{d}x = \quad = \gamma_j \gamma_k \frac{\alpha}{\sqrt{\pi}} \frac{1}{\beta\alpha} \int_{\mathbb{R}} H_{j-1}(u) H_{k-1}(u)\, \mathrm{e}^{-(2\delta^2+\alpha^2)(u/\beta\alpha)^2} \mathrm{d}u\,.$$

We have used the restriction $\alpha > 0$, and by definition $\beta > 0$, to maintain the positive orientation of the integral; if $\alpha < 0$ then the integral would have an additional "$-$" in front.

At this point, we need to simplify the term in the exponential. Substituting in the defined value of δ^2 from (4.3c) shows

$$\begin{aligned}\frac{2\delta^2 + \alpha^2}{(\beta\alpha)^2} &= \frac{2\frac{\alpha^2}{2}(\beta^2 - 1) + \alpha^2}{(\beta\alpha)^2} \\ &= \frac{\alpha^2\beta^2 - \alpha^2 + \alpha^2}{(\beta\alpha)^2} \\ &= 1.\end{aligned}$$

The orthogonality integral then becomes

$$\int_{\mathbb{R}} \varphi_j(x)\varphi_k(x)\rho(x)\mathrm{d}x = \frac{\gamma_j\gamma_k}{\beta\sqrt{\pi}} \int_{\mathbb{R}} H_{j-1}(u)H_{k-1}(u)\,\mathrm{e}^{-u^2}\mathrm{d}u\,.$$

Using (4.5) we can replace the integral with

$$\int_{\mathbb{R}} \varphi_j(x)\varphi_k(x)\rho(x)\mathrm{d}x = \frac{\gamma_j\gamma_k}{\beta\sqrt{\pi}}\sqrt{\pi}\,2^{j-1}\Gamma(j)\delta_{j,\,k}\,.$$

At this point, we have the necessary orthogonality when $j \neq k$ because $\delta_{j,\,k} = 0$. If $j = k$, the integral simplifies to

$$\int_{\mathbb{R}} \varphi_j(x)^2\rho(x)\mathrm{d}x = \gamma_j^2\frac{2^{j-1}\Gamma(j)}{\beta}\,.$$

Substituting in γ_j from (4.3b) yields

$$\begin{aligned}\int_{\mathbb{R}} \varphi_j(x)^2\rho(x)\mathrm{d}x &= \left(\sqrt{\frac{\beta}{2^{j-1}\Gamma(j)}}\right)^2 \frac{2^{j-1}\Gamma(j)}{\beta} \\ &= 1\,,\end{aligned}$$

which is the desired result.

4.3.2 Multivariate eigenfunction expansion

As mentioned earlier, the multivariate case is easily obtained using the tensor product form of the Gaussian kernel, i.e., for d-variate functions we have

$$\begin{aligned}K(\boldsymbol{x},\boldsymbol{z}) &= \exp\left(-\varepsilon_1^2(x_1-z_1)^2 - \ldots - \varepsilon_d^2(x_d-z_d)^2\right) \\ &= \prod_{j=1}^{d}\exp\left(-\varepsilon_j^2(x_j-z_j)^2\right) \\ &= \prod_{j=1}^{d}\sum_{m=1}^{\infty}\lambda_m\varphi_m(x)\varphi_m(z) \\ &= \sum_{\boldsymbol{m}\in\mathbb{N}^d}\lambda_{\boldsymbol{m}}\varphi_{\boldsymbol{m}}(x)\varphi_{\boldsymbol{m}}(z),\end{aligned}$$

where

$$\lambda_{\boldsymbol{m}} = \prod_{j=1}^{d} \lambda_{(\boldsymbol{m})_j} = \prod_{j=1}^{d} \sqrt{\frac{\alpha_j^2}{\delta_j^2 + \alpha_j^2 + \varepsilon_j^2}} \left(\frac{\varepsilon_j^2}{\delta_j^2 + \alpha_j^2 + \varepsilon_j^2} \right)^{(\boldsymbol{m})_j - 1}, \tag{4.6a}$$

$$\varphi_{\boldsymbol{m}}(\boldsymbol{x}) = \prod_{j=1}^{d} \varphi_{(\boldsymbol{m})_j}(x_j) = \prod_{j=1}^{d} \gamma_{(\boldsymbol{m})_j} \exp\left(-\delta_j^2 x_j^2\right) H_{(\boldsymbol{m})_j - 1}(\beta_j \alpha_j x_j), \tag{4.6b}$$

and $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$. The multiindex $\boldsymbol{m}$ has d components, of which the jth is referenced by $(\boldsymbol{m})_j$. This is done to distinguish referencing within a multiindex, or between several multiindices, e.g., $(\boldsymbol{m}_1)_j$ and $(\boldsymbol{m}_2)_j$.

Note that here we are allowed to take different shape parameters ε_j for different space dimensions (i.e., K would be an anisotropic kernel), or we can take them all equal, i.e., $\varepsilon_j = \varepsilon$, $j = 1, \ldots, d$ (and then K is isotropic or radial). Because ε_j is allowed to vary by dimension, α_j may also vary by dimension. For the purposes of this work, we restrict ourselves to using the same ε_j in all dimensions (i.e., $\varepsilon_j = \varepsilon$ for $j = 1, \ldots, d$), but future work will investigate the use of individual ε_j for each dimension. This is significant in attempting to achieve predictable, dimension-independent rates of convergence [61].

4.4 A stable evaluation algorithm

The starting point in [70] was an expansion of the form

$$\mathrm{e}^{-\varepsilon^2(x-z)^2} = \sum_{m=0}^{\infty} \frac{(2\varepsilon^2)^m}{m!} x^m \mathrm{e}^{-\varepsilon^2 x^2} z^m \mathrm{e}^{-\varepsilon^2 z^2}. \tag{4.7}$$

However, the authors claimed that this series is not ideal for stable "flat" limit calculations since it does not provide an effective separation of the terms that cause the ill-conditioning associated with small ε-values. Most likely, the poor conditioning of this new basis is due to the fact that the functions $x \mapsto x^m \mathrm{e}^{-\varepsilon^2 x^2}$ are not

orthogonal in $L_2(\mathbb{R})$. Indeed, for $\varepsilon \to 0$ these functions converge to the standard monomial basis giving rise to the notoriously ill-conditioned Vandermonde matrix. Therefore, the authors followed up their initial expansion with a transformation to polar coordinates and an expansion in terms of Chebyshev polynomials.

If one instead uses an eigenfunction expansion as discussed in the previous section, then the source of ill-conditioning of the Gaussian basis can be separated from the eigenfunctions and moved into the eigenvalues. Moreover, for a smooth kernel such as the Gaussian the eigenvalues decay very quickly so that we should now be able to directly (i.e., without having to deal with an additional transformation to Chebyshev polynomials) follow the QR-based strategy suggested in [70].

4.4.1 The RBF-QR algorithm

In particular, we now use the Gaussian kernel (4.1) along with its eigenvalues (4.6a) and eigenfunctions (4.6b) as discussed above. To keep the notation simple, we assume that the eigenvalues and their associated eigenfunctions have been sorted linearly so that we can enumerate them with integer subscripts instead of the multi-index notation used in (4.6a-4.6b). This matter is not a trivial one and needs to be dealt with carefully in the implementation. The QR-based algorithm of [70] corresponds to the following: using the eigen-decomposition of the kernel function K, we can rewrite the kernel matrix K appearing in the linear system for

the interpolation problem as

$$\mathsf{K} = \begin{pmatrix} K(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & K(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & & \vdots \\ K(\boldsymbol{x}_N, \boldsymbol{x}_1) & \dots & K(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$
$$= \begin{pmatrix} \varphi_1(\boldsymbol{x}_1) & \dots & \varphi_M(\boldsymbol{x}_1) & \dots \\ \vdots & & \vdots & \\ \varphi_1(\boldsymbol{x}_N) & \dots & \varphi_M(\boldsymbol{x}_N) & \dots \end{pmatrix} \begin{pmatrix} \lambda_1 & & & \\ & \ddots & & \\ & & \lambda_M & \\ & & & \ddots \end{pmatrix} \begin{pmatrix} \varphi_1(\boldsymbol{x}_1) & \dots & \varphi_1(\boldsymbol{x}_N) \\ \vdots & & \vdots \\ \varphi_M(\boldsymbol{x}_1) & \dots & \varphi_M(\boldsymbol{x}_N) \\ \vdots & & \vdots \end{pmatrix}.$$

Of course we can not conduct computation on infinite matrices, so we must choose a truncation value M after which we neglect the remaining terms in the series. Since the eigenvalues $\lambda_m \to 0$ as $m \to \infty$ we have a necessary condition to encourage such a truncation. A particular choice of M is discussed in Section 4.6, but given that we have chosen one the system changes to the much more manageable

$$\mathsf{K} = \underbrace{\begin{pmatrix} \varphi_1(\boldsymbol{x}_1) & \dots & \varphi_M(\boldsymbol{x}_1) \\ \vdots & & \vdots \\ \varphi_1(\boldsymbol{x}_N) & \dots & \varphi_M(\boldsymbol{x}_N) \end{pmatrix}}_{=\Phi} \underbrace{\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_M \end{pmatrix}}_{=\Lambda} \underbrace{\begin{pmatrix} \varphi_1(\boldsymbol{x}_1) & \dots & \varphi_1(\boldsymbol{x}_N) \\ \vdots & & \vdots \\ \varphi_M(\boldsymbol{x}_1) & \dots & \varphi_M(\boldsymbol{x}_N) \end{pmatrix}}_{=\Phi^T},$$

or simply

$$\mathsf{K} = \Phi \Lambda \Phi^T. \tag{4.8}$$

Although our specific choice of M is postponed until later, it is important to note that since it is our immediate goal to avoid the ill-conditioning associated with radial basis interpolation as $\varepsilon \to 0$, we require $M \geq N$. This is in accordance with the work of Fornberg, and seeks to ensure that all of the eigenfunctions φ_m, $m = 1, \dots, M$, used above are obtained to machine precision. This also justifies — for all practical computations — our continued use of an equality sign for the matrix factorization of K.

We are interested in determining a new basis where the interpolation can be conducted without the condition issues associated with the matrix K, while still remaining in the same space spanned by the Gaussian kernel function K. Thus an invertible matrix X^{-1} is needed such that $\mathsf{K}\mathsf{X}^{-1}$ is better conditioned than K. Of course, the simple choice would be $\mathsf{X}^{-1} = \mathsf{K}^{-1}$, but if that were available to machine precision this problem would be trivial.

The structure of the matrix $\mathsf{\Phi}$ provides one possible avenue since its mth column contains only values of the mth eigenfunction at all the data sites $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. This provides the opportunity to conduct a QR decomposition of $\mathsf{\Phi}$ without mixing eigenfunctions of different orders. For $M > N$, the matrix $\mathsf{\Phi}$ is "short and fat", meaning that the QR decomposition takes the form

$$\begin{pmatrix} \varphi_1(\boldsymbol{x}_1) & \ldots & \varphi_N(\boldsymbol{x}_1) & | & \varphi_{N+1}(\boldsymbol{x}_1) & \ldots & \varphi_M(\boldsymbol{x}_1) \\ \vdots & & \vdots & | & \vdots & & \vdots \\ \varphi_1(\boldsymbol{x}_N) & \ldots & \varphi_N(\boldsymbol{x}_N) & | & \varphi_{N+1}(\boldsymbol{x}_N) & \ldots & \varphi_M(\boldsymbol{x}_N) \end{pmatrix} = \mathsf{Q} \begin{pmatrix} & | & \\ \mathsf{R}_1 & | & \mathsf{R}_2 \\ & | & \end{pmatrix},$$

where the R_1 block is a square matrix of size N and R_2 is $N \times (M - N)$.

Substituting this decomposition for $\mathsf{\Phi}^T$ in (4.8) we see

$$\mathsf{K} = \mathsf{\Phi}\mathsf{\Lambda}\mathsf{R}^T\mathsf{Q}^T.$$

By imposing the same block structure on $\mathsf{\Lambda}$ that was imposed on R we can rewrite

this full system in blocks as

$$\begin{aligned}
\mathsf{K} &= \Phi \begin{pmatrix} \Lambda_1 & \\ & \Lambda_2 \end{pmatrix} \begin{pmatrix} \mathsf{R}_1^T \\ \mathsf{R}_2^T \end{pmatrix} \mathsf{Q}^T \\
&= \Phi \begin{pmatrix} \Lambda_1 \mathsf{R}_1^T \\ \Lambda_2 \mathsf{R}_2^T \end{pmatrix} \mathsf{Q}^T \\
&= \Phi \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix} \Lambda_1 \mathsf{R}_1^T \mathsf{Q}^T \,. \qquad (4.9)
\end{aligned}$$

The final form of this representation is significant because of the structure of the $\Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1}$ term. In Section 4.3 we noticed that the eigenvalues $\lambda_m \to 0$ as $m \to \infty$ (and especially quickly if ε^2 is small relative to $\delta^2 + \alpha^2$). This means that the eigenvalues in Λ_2 are smaller than those in Λ_1 and thus none of the entries in $\mathsf{R}_2^T \mathsf{R}_1^{-T}$ are magnified when we form $\Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1}$.

Since we can perform the multiplications by the diagonal matrices Λ_2 and Λ_1^{-1} *analytically* we avoid the ill-conditioning that would otherwise be associated with underflow (the values in Λ_2 are as small as ε^{2M-2}) or overflow (the values in Λ_1^{-1} are as large as ε^{-2N-2}).

Let us now return to the original goal of determining a new basis that allows us to conduct the interpolation in a safe and stable manner. Since we have now concluded that as $\varepsilon \to 0$ the $\Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1}$ term poses no problems, we are left to consider the $\Lambda_1 \mathsf{R}_1^T \mathsf{Q}^T$ term. This matrix is nonsingular if Φ^T has full row rank because Λ_1 is diagonal with nonzero (in exact arithmetic) values and R_1 is upper triangular and has the same rank as Φ^T. Because of the orthogonality of the eigenfunctions φ_m, $m = 1, \ldots, M$, we have nonsingularity of R_1 and thus a good

choice for the matrix X is given by

$$\mathsf{X} = \Lambda_1 \mathsf{R}_1^T \mathsf{Q}^T. \tag{4.10}$$

We are now interested in the new system defined by

$$\begin{aligned}
\Psi &= \mathsf{K}\mathsf{X}^{-1} \\
&= \left[\Phi \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix} \Lambda_1 \mathsf{R}_1^T \mathsf{Q}^T \right] \left[\Lambda_1 \mathsf{R}_1^T \mathsf{Q}^T \right]^{-1} \\
&= \Phi \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix}.
\end{aligned} \tag{4.11}$$

Here we used (4.10) and the decomposition (4.9) of K.

We can interpret the columns of Ψ as being created by new basis functions which can be thought of as the first N eigenfunctions plus a correction involving a linear combination of the next $M - N$ eigenfunctions:

$$\begin{aligned}
\Psi &= \begin{pmatrix} \varphi_1(\boldsymbol{x}_1) & \dots & \varphi_N(\boldsymbol{x}_1) & | & \varphi_{N+1}(\boldsymbol{x}_1) & \dots & \varphi_M(\boldsymbol{x}_1) \\ \vdots & & \vdots & | & \vdots & & \vdots \\ \varphi_1(\boldsymbol{x}_N) & \dots & \varphi_N(\boldsymbol{x}_1) & | & \varphi_{N+1}(\boldsymbol{x}_1) & \dots & \varphi_M(\boldsymbol{x}_N) \end{pmatrix} \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix} \\
&= \left(\begin{array}{c|c|c} & & \\ \Phi_1 & | & \Phi_2 \\ & & \end{array} \right) \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix} \\
&= \Phi_1 + \Phi_2 \left[\Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \right].
\end{aligned} \tag{4.12}$$

In order to see the actual basis functions we consider the vector $\Psi(\boldsymbol{x})$ defined as

$$\begin{aligned}
\Psi(\boldsymbol{x})^T &= \begin{pmatrix} \psi_1(\boldsymbol{x}) & \dots & \psi_N(\boldsymbol{x}) \end{pmatrix} \\
&= \begin{pmatrix} \varphi_1(\boldsymbol{x}) & \dots & \varphi_M(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix}.
\end{aligned}$$

This representation is in the same fashion that the standard kernel basis could be written as

$$\begin{aligned}\boldsymbol{k}(\boldsymbol{x})^T &= \begin{pmatrix} K(\boldsymbol{x},\boldsymbol{x}_1) & \dots & K(\boldsymbol{x},\boldsymbol{x}_N) \end{pmatrix} \\ &= \begin{pmatrix} \varphi_1(\boldsymbol{x}) & \dots & \varphi_M(\boldsymbol{x}) \end{pmatrix} \Lambda \Phi^T \\ &= \begin{pmatrix} \varphi_1(\boldsymbol{x}) & \dots & \varphi_M(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix} \Lambda_1 \mathsf{R}_1^T \mathsf{Q}^T,\end{aligned} \tag{4.13}$$

only now the ill-conditioning related to Λ_1 has been removed from the basis.

It is tempting to think of this as a preconditioning technique, as our goal of producing a well conditioned matrix $\mathsf{K}\mathsf{X}^{-1}$ is the same as a preconditioner in an iterative method. In reality, X is not a preconditioner, because we are no longer interested in solving the original system. Instead, X^{-1} is applied as a linear transformation, or a change of basis, to allow us to describe the interpolation system in a stable basis rather than the ill-conditioned Gaussian basis. The linear system is different than the original linear system, but because the transformation matrix X is nonsingular, the interpolants span the same function space.

The approach described in this section should be applicable whenever one knows the eigenfunction (or other orthonormal basis) expansion of a positive definite kernel. One such example is provided by the approach taken in [72] for stable radial basis function approximation on the sphere, where the connection between the (zonal) kernels being employed on the sphere and spherical harmonics, which are the eigenfunctions of the Laplace-Beltrami operator on the sphere, has traditionally been a much closer one (see, e.g., [64]).

4.4.2 Implementation details

The interpolation problem described in Section 4.2 can be written in matrix notation as

$$\mathsf{K}\boldsymbol{c} = \boldsymbol{y}, \tag{4.14}$$

where K is the $N \times N$ kernel matrix, $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ is input data denoting the function values to be fitted at the points $\boldsymbol{x}_n$, $n = 1, \ldots, N$, and $\boldsymbol{c}$ is the unknown vector of coefficients. In the new basis $\Psi = (\psi_1, \ldots, \psi_N)^T$ the system is still of size $N \times N$ and can be written in the form

$$\Psi\boldsymbol{b} = \boldsymbol{y},$$

where the matrix Ψ was defined in (4.11), $\boldsymbol{y}$ is as above, and $\boldsymbol{b}$ is a new vector of coefficients. Once we have solved for these coefficients, the Gaussian kernel interpolant s can be evaluated at an arbitrary point $\boldsymbol{x} \in \mathbb{R}^d$ via

$$s(\boldsymbol{x}) = \Psi(\boldsymbol{x})^T\boldsymbol{b}.$$

Using (4.11), the linear solve itself takes the form

$$\Phi \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix} \boldsymbol{b} = \boldsymbol{y}, \tag{4.15}$$

where as before

$$\Phi^T = \begin{pmatrix} \mathsf{R}_1^T \\ \mathsf{R}_2^T \end{pmatrix} \mathsf{Q}^T,$$

and the block structure is defined with first blocks of size $N \times N$,

$$\Lambda = \begin{pmatrix} \Lambda_1 & \\ & \Lambda_2 \end{pmatrix}, \qquad \Phi = \begin{pmatrix} \Phi_1 & \Phi_2 \end{pmatrix}.$$

At this point, the system (4.15) could be solved by conducting the matrix-matrix multiplication and working with the resulting $N \times N$ matrix:

$$\left[\Phi_1 + \Phi_2 \left[\Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1}\right]\right] \boldsymbol{b} = \boldsymbol{y}.$$

Doing so, however, would disregard the QR decomposition that was already computed. Instead, we can use the fact that $\Phi = \mathsf{QR}$ in (4.15) to rewrite the system as

$$\begin{aligned} & \mathsf{Q} \begin{pmatrix} \mathsf{R}_1 & \mathsf{R}_2 \end{pmatrix} \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix} \boldsymbol{b} = \boldsymbol{y} \\ \iff \quad & \mathsf{QR}_1 \begin{pmatrix} \mathsf{I}_N & \mathsf{R}_1^{-1}\mathsf{R}_2 \end{pmatrix} \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1} \end{pmatrix} \boldsymbol{b} = \boldsymbol{y} \\ \iff \quad & \mathsf{QR}_1 \left[\mathsf{I}_N + \mathsf{R}_1^{-1}\mathsf{R}_2 \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1}\right] \boldsymbol{b} = \boldsymbol{y}\,. \end{aligned}$$

This is especially nice because the term $\mathsf{R}_1^{-1}\mathsf{R}_2$ is just the transpose of $\mathsf{R}_2^T \mathsf{R}_1^{-T}$ and thus its value can be saved from earlier and the cost of $\mathcal{O}(N^2(M-N))$ can be saved.

Now the linear solve can be broken into two parts, where

$$\mathsf{QR}_1 \hat{\boldsymbol{b}} = \boldsymbol{y}, \tag{4.16a}$$

$$\left[\mathsf{I}_N + \mathsf{R}_1^{-1}\mathsf{R}_2 \Lambda_2 \mathsf{R}_2^T \mathsf{R}_1^{-T} \Lambda_1^{-1}\right] \boldsymbol{b} = \hat{\boldsymbol{b}}. \tag{4.16b}$$

Solving (4.16a) is almost trivial, since Q is orthonormal and R_1 is upper triangular. Solving (4.16b) can be done cleverly depending on the value of M:

- If M is chosen such that $M < 2N$, then the linear system can be treated as a low rank update to the identity and the inverse can be applied with the Sherman-Morrison formula. Total cost would be $\mathcal{O}\left((N^2(M-N))\right)$.

- If $M \geq 2N$, the cost of the interior inverse in the Sherman-Morrison formula would be greater than simply solving the original system, so a direct approach is preferred. Total cost would be $\mathcal{O}(N^3)$.

Because this search for a new basis is conducted with the goal of working in the "flat" kernel regime, it is logical to assume that we are dealing with small ε. Therefore, it is reasonable to assume that the value of M can be chosen relatively close to N because additional terms would be truncated. As a result, (4.16b) could be solved using the Sherman-Morrison formula:

$$\begin{aligned}\boldsymbol{b} &= \left[\mathsf{I}_N + \mathsf{R}_1^{-1}\mathsf{R}_2\mathsf{\Lambda}_2\mathsf{R}_2^T\mathsf{R}_1^{-T}\mathsf{\Lambda}_1^{-1}\right]^{-1}\hat{\boldsymbol{b}} \\ &= \left[\mathsf{I}_N - \mathsf{R}_1^{-1}\mathsf{R}_2\left[\mathsf{I}_{M-N} + \mathsf{\Lambda}_2\mathsf{R}_2^T\mathsf{R}_1^{-T}\mathsf{\Lambda}_1^{-1}\mathsf{R}_1^{-1}\mathsf{R}_2\right]^{-1}\mathsf{\Lambda}_2\mathsf{R}_2^T\mathsf{R}_1^{-T}\mathsf{\Lambda}_1^{-1}\right]\hat{\boldsymbol{b}}\,.\end{aligned}$$

In some circumstances it may be preferable to compute $\mathsf{\Psi}$ and invert it directly, in lieu of using Q and R_1 separately. Because this chapter is concerned with the stability of the solution method, not the speed with which it is computed, we always compute $\mathsf{\Psi}$ and solve $\mathsf{\Psi}\boldsymbol{b} = \boldsymbol{y}$. In the future, we hope to study the condition of each of the solution options and study their practicality.

Choosing the length of the eigenfunction expansion

Thus far the value of M has been fixed but left unknown. Our choice of M coincides with that of [72], where M is the smallest value that satisfies $\lambda_M < \epsilon_{mach}\lambda_N$. ϵ_{mach} is machine precision (assumed to be 10^{-16} and λ_m is defined in (4.4b)). Solving

that inequality for M produces

$$\begin{aligned}
\sqrt{\frac{\alpha^2}{\delta^2+\alpha^2+\varepsilon^2}}\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right)^{M-1} &< \epsilon_{mach}\sqrt{\frac{\alpha^2}{\delta^2+\alpha^2+\varepsilon^2}}\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right)^{N-1} \\
\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right)^{M-N} &< \epsilon_{mach} \\
(M-N)\log\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right) &< \log(\epsilon_{mach}) \\
M &> N + \log(\epsilon_{mach})\left(\log\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right)\right)^{-1}.
\end{aligned} \tag{4.17}$$

This bound is derived in 1D, although it extends naturally to multiple dimensions. In doing so, the uniqueness of the eigenfunction expansion is lost because there may be several multiindices $\boldsymbol{M}$ which satisfy the inequality. Rederiving this for a d dimensional problem using $|\boldsymbol{M}| = \sum_j M_j$ and (4.6a) produces

$$\begin{aligned}
\left(\frac{\alpha^2}{\delta^2+\alpha^2+\varepsilon^2}\right)^{d/2}\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right)^{|\boldsymbol{M}|-d} &< \epsilon_{mach}\left(\frac{\alpha^2}{\delta^2+\alpha^2+\varepsilon^2}\right)^{d/2}\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right)^{|\boldsymbol{N}|-d} \\
\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right)^{|\boldsymbol{M}|-|\boldsymbol{N}|} &< \epsilon_{mach} \\
|\boldsymbol{M}| &> |\boldsymbol{N}| + \log(\epsilon_{mach})\left(\log\left(\frac{\varepsilon^2}{\delta^2+\alpha^2+\varepsilon^2}\right)\right)^{-1}.
\end{aligned} \tag{4.18}$$

For a simple example, suppose $d = 2$, $\mathrm{e}^2/(\delta^2+\alpha^2+\varepsilon^2) = e^{-1}$, $N = 5$, $\epsilon_{mach} = 10^{-16}$. That makes the inequality

$$|\boldsymbol{M}| > |\boldsymbol{N}| + 16.$$

When trying to determine $\boldsymbol{N}$ to separate the eigenfunctions from the correction we need to consider the leading eigenvalue indices:

$$\begin{aligned}
N_1 &= 1 \;\; 2 \;\; 1 \;\; \mathbf{3} \;\; \mathbf{2}^* \;\; \mathbf{1} \;\; 4 \;\; ... \\
N_2 &= 1 \;\; 1 \;\; 2 \;\; \mathbf{1} \;\; \mathbf{2}^* \;\; \mathbf{3} \;\; 1 \;\; ...
\end{aligned}$$

The entries that are starred are in the fifth index so they are included in the Φ_1 matrix and any higher terms are pushed to the correction. Those that are in bold are all of the same magnitude as the fifth index and could just as easily have been chosen for the Φ_1 matrix instead. This exposes the nonuniqueness, but we don't believe this to be a problem thanks to the nonuniqueness results described in [70].

Because $N = 5$ for this problem we know $|\boldsymbol{N}| = 4$, and thus to meet the truncation criteria we would need $|\boldsymbol{M}| > 20$. Of course, thanks to Pascal's Triangle, we know that there are 19 different values which satisfy $|\boldsymbol{M}| = 20$ meaning that the number of eigenfunctions needed in the correction Φ_2 may be quite significant.

Stability and numerical concers

There are numerous technical details which need to be addressed before this Gaussian eigenfunction solution via RBF-QR can be successfully implemented. The first of these points deals with the computation of δ^2: when $\beta^2 \approx 1$, the computation of

$$\delta^2 = \frac{\alpha^2}{2}(\beta^2 - 1)$$

suffers from cancelation error. We can alleviate this problem by considering the structure of β^2,

$$\beta^2 = \sqrt{1 + 4\left(\frac{\varepsilon}{\alpha}\right)^2}.$$

From the definition, we realize that $\beta^2 \to 1$ as $\frac{\varepsilon}{\alpha} \to 0$, and we can expand β^2 in a series around that point,

$$\beta^2 = 1 + 2\left(\frac{\varepsilon}{\alpha}\right)^2 - 2\left(\frac{\varepsilon}{\alpha}\right)^4 + \mathcal{O}\left(\left(\frac{\varepsilon}{\alpha}\right)^6\right).$$

Using this, we see that δ^2 can be safely approximated for $\varepsilon \to 0$ or $\alpha \to \infty$ by using the series

$$\begin{aligned}\delta^2 &\approx \frac{\alpha^2}{2}\left(1 + 2\left(\frac{\varepsilon}{\alpha}\right)^2 - 2\left(\frac{\varepsilon}{\alpha}\right)^4 - 1\right) \\ &\approx \varepsilon^2 - \frac{\varepsilon^4}{\alpha^2}.\end{aligned}$$

This series expansion also resolves the indeterminate form for δ^2 which would arise for $\alpha \to \infty$ and $\beta^2 - 1 \to 0$.

For large values of m, evaluating $\varphi_m(x)$ is problematic because each of the components of the eigenfunctions are potentially of very large scale. The actual eigenfunctions themselves may be of a reasonable size, but each of the components of the eigenfunctions may experience overflow or underflow if evaluated individually. Because the eigenfunctions are formed by a product of 3 components (γ_m, $\mathrm{e}^{-\delta^2 x^2}$ and $H_{m-1}(\beta\alpha x)$) we can safely compute $\varphi_m(x)$ using logarithms:

$$\begin{aligned}\log \varphi_m(x) &= \log \gamma_m - \delta^2 x^2 + \log H_{m-1}(\beta\alpha x). \\ &= \frac{1}{2}\left(\log \beta - (m-1)\log 2 - \log \Gamma(m)\right) - \delta^2 x^2 + \log H_{m-1}(\beta\alpha x).\end{aligned}$$

This form is useful because $\log \Gamma(m)$ can be computed directly, but we must still consider the $\log H_{m-1}(\beta\alpha x)$. There is a worry about the fact that $H_{m-1}(\beta\alpha x)$ can be negative, but taking the log of that here is not a problem in exact arithmetic because when the exponential is applied to recover $\varphi_m(x)$ the imaginary term drops out.

The more serious concern is that for large arguments, or large indices, this polynomial is very large, which means that an asymptotic form of it must be found for safe computation. Referring to [1], we see that three different regions need to be considered:

- $|\beta\alpha x| \ll \sqrt{2m}$ - The inner region has the asymptotic expansion

$$\begin{aligned} H_{m-1}(\beta\alpha x) \approx & \sqrt{2}(1-\xi^2)^{1/4} \cdot \\ & \exp\left(m/2(\log(2m)) - 1 + 2\xi^2)\right) \cdot \\ & \cos\left(m(\xi\sqrt{1-\xi^2} - \pi/2) + (m+1/2)\sin^{-1}(\xi)\right), \end{aligned}$$

  where $\xi = \beta\alpha x/\sqrt{2n}$.

- $|\beta\alpha x| \approx \sqrt{2m}$ - The transition region has the asymptotic expansion

$$\begin{aligned} H_{m-1}(\beta\alpha x) \approx & \sqrt{2\pi} m^{1/6} \exp\left(m/2\log(2m) - 3m/2\right) \cdot \\ & \exp\left(\sqrt{2n}\beta\alpha x\right) \cdot \\ & \mathrm{Ai}\left(\sqrt{2}m^{1/6}(\beta\alpha x - \sqrt{2m})\right), \end{aligned}$$

  where Ai is the Airy function. It should be noted that $\mathrm{Ai}(x)$ itself actually needs the asymptotic expansion $\exp(-2x^{3/2}/3)/(2\sqrt{\pi}x^{1/4})$ when x is too large.

- $|\beta\alpha x| \gg \sqrt{2m}$ - The outer region has the asymptotic expansion

$$\begin{aligned} H_{m-1}(\beta\alpha x) \approx & \sqrt{(1+x/\zeta)/2} \cdot \\ & \exp\left((x^2 - \zeta x - m)/2 + m\log(\zeta + x)\right), \end{aligned}$$

  where $\zeta = \sqrt{x^2 - 2m}$.

For each of these regions, the sign of the output is determined before the logarithm is applied to make sure that complex terms do not arise during the computation.

It is possible that some of these concerns regarding domain of evaluation can be neglected by a skillful rescaling of the input data. More research needs to be performed before a statement can be made, given the interaction of the α term with the domain of the data.

4.5 Numerical experiments for interpolation

To determine the eigenfunction expansion's ability to accurately carry out radial basis interpolation with Gaussian kernels in the $\varepsilon \to 0$ limit, some experiments need to be conducted. This computation was conducted in MATLAB, using the built-in QR factorization and triangular solvers for the RBF-QR results. A MATLAB implementation of a direct solver via the "backslash" operator \ was used to produce the RBF-Direct results for comparison. The polynomial fits were generated with the MATLAB function `polyfit`. The first part of this section demonstrates the accuracy of the stable basis in reaching the $\varepsilon \to 0$ limit without ill-conditioning. In the second part we identify issues related to the α parameter which need to be addressed for larger problems.

4.5.1 1D and 2D interpolation

The first set of experiments is limited to 1D and studies the effect of increasing the number of data points N. All the data points are located at the Chebyshev nodes within an interval $[x_a, x_b]$

$$x_i = \frac{1}{2}(x_b + x_a) - \frac{1}{2}(x_b - x_a)\cos\left(\pi \frac{i-1}{N-1}\right), \qquad i = 1, \ldots, N. \tag{4.19}$$

The interpolation is conducted using the eigenfunction-QR algorithm (abbreviated RBF-QR) over shape parameter values logarithmically spaced within $\varepsilon \in [10^{-2}, 10^{0.4}]$. For comparison, the solution to (4.14) is computed using the traditional [60] RBF solution (abbreviated RBF-Direct) over $\varepsilon \in [10^{-2}, 10^{1}]$.

Input values are produced by a function f and the error in the interpolant s is

computed by

$$\text{error} = \frac{1}{\bar{N}} \frac{1}{\|f(\bar{\boldsymbol{x}})\|_2} \sqrt{\sum_{k=1}^{\bar{N}} (f(\bar{x}_k) - s(\bar{x}_k))^2}, \tag{4.20}$$

where $\bar{x}_k$ are $\bar{N}$ uniformly spaced points at which s is compared to f. The term

$$\|f(\bar{\boldsymbol{x}})\|_2 = \sqrt{\sum_{k=1}^{N} f(\bar{x}_k)^2}$$

accounts for the magnitude of the function on the domain. For the 1D experiments, $\bar{N} = 1000$, although this choice was made arbitrarily.

The experiments in 1D can be seen in Figure 4.1. Two functions were considered, first

$$f_1(x) = \frac{\sinh x}{1 + \cosh x}, \qquad x \in [-3, 3],$$

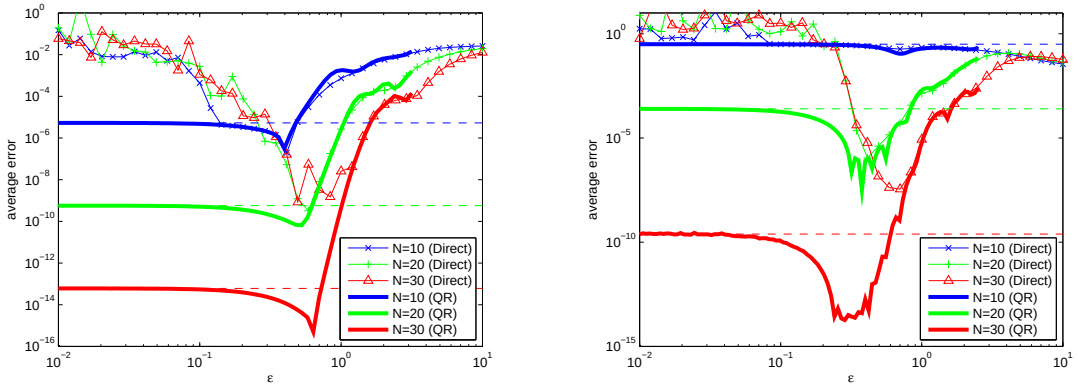
using $N = \{10, 20, 30\}$ and $\alpha = 1$, which can be seen in Figure 4.1a. The second function considered was

$$f_2(x) = \sin\left(\frac{x}{2}\right) - 2\cos x + 4\sin(\pi x), \qquad x \in [-4, 4],$$

using $N = \{10, 20, 30\}$ and $\alpha = .65$, which can be seen in Figure 4.1b.

These initial results confirm that for $\varepsilon \to 0$ the RBF-QR algorithm evaluates the Gaussian interpolant without the ill-conditioning associated with RBF-Direct. Note that two different choices of α were used for these two problems - more on this to follow.

The examples presented here also illustrate that interpolation with Gaussian kernels is more accurate than polynomial interpolation (which corresponds to the $\varepsilon \to 0$ limit) — even though both methods are known to be spectrally accurate. The errors for the corresponding polynomial interpolants are included as dashed horizontal lines in Figure 4.1.



(a) $f_1(x) = \sinh x(1 + \cosh x)^{-1}$ (b) $f_2(x) = \sin\left(\frac{x}{2}\right) - 2\cos x + 4\sin(\pi x)$

Figure 4.1: Comparison of RBF-QR and RBF-Direct; dashed horizontal lines represent errors of limiting polynomial interpolants.

See Figure 4.2 for the interpolation results of a small 2D problem involving the function

$$f_4(x, y) = \cos(x^2 + y^2), \qquad (x, y) \in [-3, 3]^2,$$

sampled at the aforementioned Chebyshev nodes. The RBF-QR scheme works as

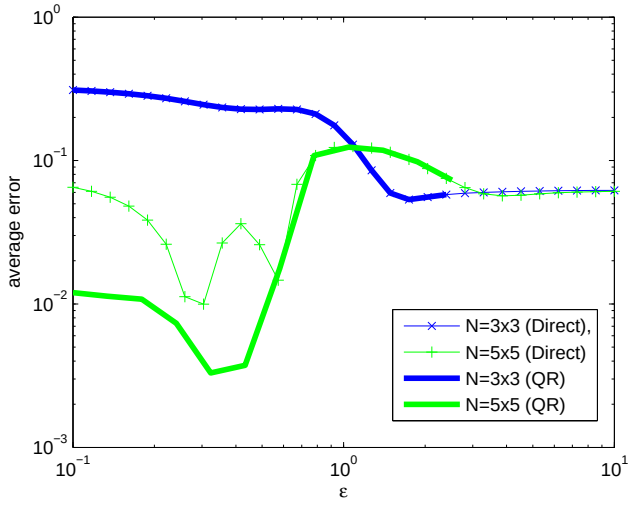


Figure 4.2: RBF-QR is able to resolve the interpolant accurately as $\varepsilon \to 0$

it should, but the computational cost is quite significant, which is why fewer ε values were considered than in the previous 1D graphs. This issue is discussed

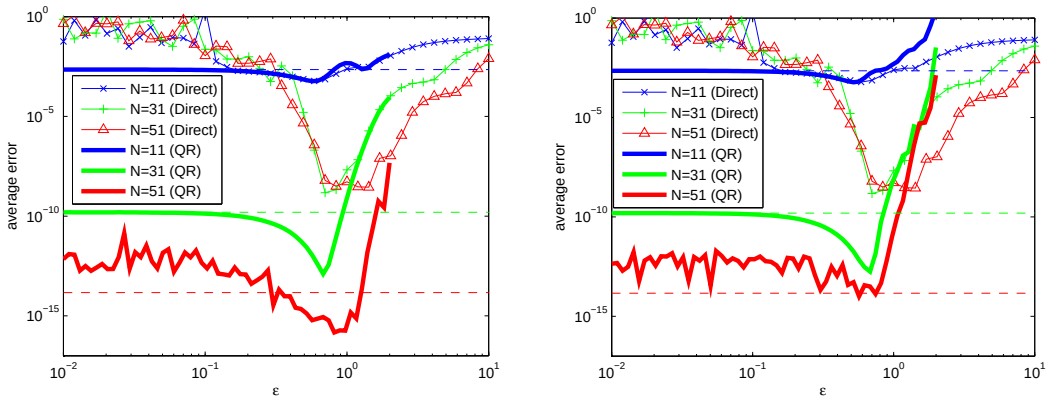
further in Section 4.6.

4.5.2 Complications for the interpolation algorithm

Although the previous experiments successfully illustrate the usefulness of the eigenfunction expansion for the solution of the interpolation problem, performing a similar test with the function

$$f_3(x) = 10\mathrm{e}^{-x^2} + x^2, \qquad x \in [-3, 3],$$

exposes some subtle complexities of our RBF-QR interpolation algorithm, as seen in Figure 4.3. Specific attention should be paid to the effect of increasing N on the emerging oscillations in the error of the interpolant.



(a) $\alpha = 1$ produces bad results for small ε and larger N

(b) $\alpha = 3$ produces bad results for large ε

Figure 4.3: Different α values have a significant effect on the stability of the interpolation

This new source of error is separate from the instability encountered in the $\varepsilon \to 0$ limit, although it has similar roots. Recall the structure of the eigenfunctions from earlier:

$$\varphi_m(x) = \gamma_m \exp\left(-\delta^2 x^2\right) H_{m-1}(\beta\alpha x). \tag{4.4a}$$

Much in the same way that ε is a significant source of ill-conditioning for the Gaussian basis, so the δ value (and more directly the α value) may be a source of ill-conditioning for the eigenfunction basis.

The reader may recall that the interpolation problems under consideration all exist on a compact domain, but the orthogonality properties of the eigenfunctions demand integration to infinity. The choice of α is a balancing act between interacting eigenfunctions to achieve accuracy (small α) and quickly decaying eigenfunctions to numerically maintain orthogonality (large α).

To see why, we fix α and analyze two limits

$$\begin{aligned} &\text{as } \varepsilon \to 0, \qquad \beta \to 1, \qquad \delta^2 \to \varepsilon^2, \\ &\text{as } \varepsilon \to \infty, \qquad \beta \to \sqrt{\frac{2\varepsilon}{\alpha}}, \qquad \delta^2 \to \varepsilon\alpha, \end{aligned}$$

and the effect they have on the eigenfunctions

$$\begin{aligned} \lim_{\varepsilon\to 0} \varphi_m(x) &= \gamma_m \exp(-\varepsilon^2 x^2) H_{m-1}(\alpha x) \\ \lim_{\varepsilon\to \infty} \varphi_m(x) &= \gamma_m \exp(-\varepsilon\alpha x^2) H_{m-1}(\sqrt{2\varepsilon\alpha} x) \end{aligned}$$

In the $\varepsilon \to 0$ case, the two parameters ε and α are decoupled and each can be handled according to its needs (ε for accuracy and α for orthogonality). When $\varepsilon \to \infty$ this is no longer the case, and both the exponential and polynomial portions of the eigenfunctions exist on the same scale $\sqrt{\varepsilon\alpha}x$. This is one reason why RBF-QR should not be used for larger ε.

To gain some insight on choosing α we study the eigenfunctions graphically on the domain $[-4, 4]$ in Figure 4.4. For the eigenfunctions of $\alpha = .1$, the exponential decay has not yet appeared and because of that the orthogonality is not preserved on that domain. For $\alpha = 10$, the eigenfunctions exist on drastically different

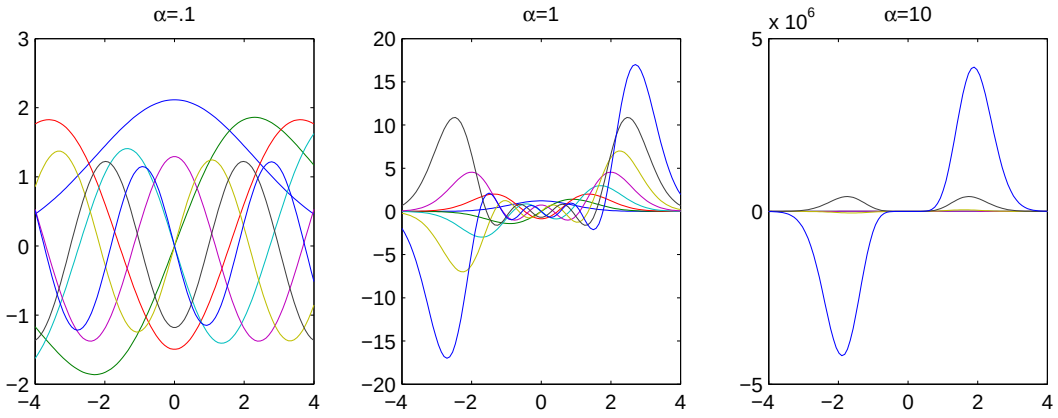


Figure 4.4: The first 8 eigenfunctions evaluated when $\varepsilon = 1$ for several α values behave very differently.

scales, meaning that effectively only the largest eigenfunctions are contributing to the solution because the smaller functions are indistinguishable from each other.

When $\alpha = 1$ there is a "good" balance of locality and distinction for the eigenfunctions. This discussion has been entirely qualitative, but it is meant to explain holistically why there is an optimal value for α to produce the true RBF interpolant for a given ε. See Figure 4.5 for computational evidence that there is an α for each ε to alleviate the instability shown in Figure 4.3.

These results show that given a set of data points x, a function f and an ε for which you want to want to produce a Gaussian interpolant there should exist an α to let you accomplish that without the $\varepsilon \to 0$ ill-conditioning. Furthermore, we see that as ε exits the asymptotically small regime, the actions of α becomes a function of ε as was predicted earlier when discussing $\varepsilon \to \infty$.

Unfortunately, these results do not describe an approach to choosing the appropriate α, they only suggest that one exists and that there may be some relationship to ε. Because this research is only interested in allowing for exploration of the $\varepsilon \to 0$ regime, we do not pursue this issue further. Determining an appropriate

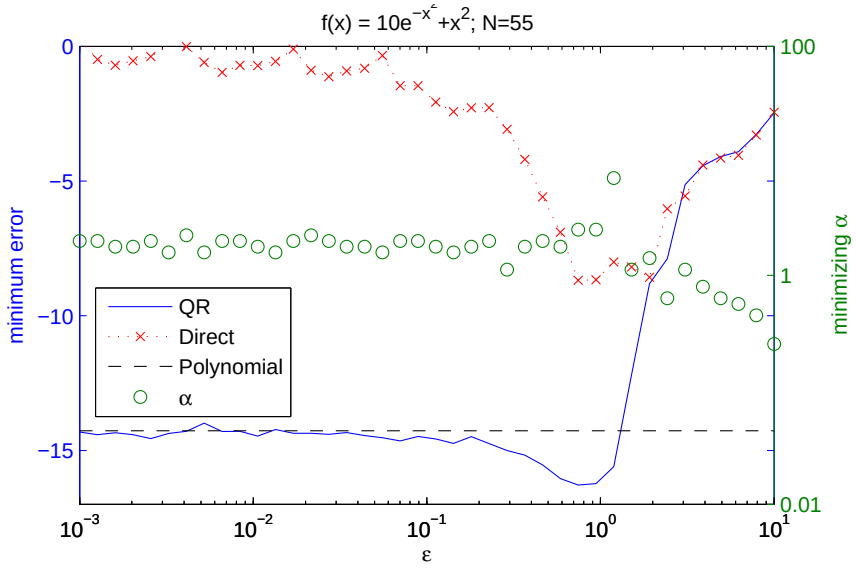


Figure 4.5: An α can be found for each ε to produce an accurate interpolant.

α a priori, and efficiently, will certainly be of significance in future work.

4.6 Early truncation

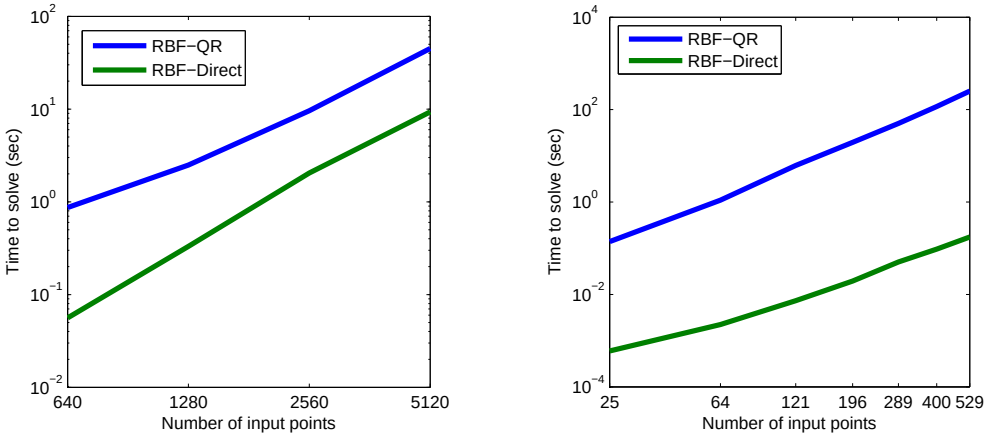
RBF-QR allows one to consider shrinking ε values for which RBF-Direct is too ill-conditioned. Unfortunately, the cost associated with RBF-QR grows substantially for increasing ε and in multiple dimensions. For larger ε RBF-Direct is a viable option, but the cost of RBF-QR in multiple dimensions is unavoidable. This is a direct result of the $\boldsymbol{M}$ definition from Section 4.4.2: assuming all the combinations of eigenfunctions are needed which satisfy (4.18), for a 2D problem the total number of columns in Φ is

$$\sum_{k=1}^{|\boldsymbol{M}|} k = \frac{|\boldsymbol{M}|(|\boldsymbol{M}|+1)}{2}$$

For comparison, this means that an $N = 25$ point 2D interpolation problem with $\varepsilon = .1$ and $\alpha = 1$ would produce $|\boldsymbol{N}| = 7$ and $|\boldsymbol{M}| = 15$ which requires 120 eigenfunctions. If $\varepsilon = 1$, $|\boldsymbol{M}| = 45$ which requires 1035 eigenfunctions to approximate

a problem of size only $N = 25$. This is obviously not a reasonable approach going forward considering the time required to perform a solution, as can be seen in Figure 4.6 for example.

In Figure 4.6 we illustrate how the cost of performing RBF-QR using the truncation strategy outlined in Section 4.4.2 is unreasonable for dimension $d = 2$ (and even more so in higher dimensions). Here cost is defined as the time required to find the coefficients associated with the interpolation; associated with this cost is the required memory for conducting the interpolation as referenced in Table 4.1a and Table 4.1b. We use a value of $\varepsilon = .1$ together with several other values of N. While the cost of doing RBF-QR in 1D is reasonable, and the payoff in terms of accuracy gain is high, this is no longer true in 2D. Clearly, we would benefit from using some other approach in higher dimensions.



(a) RBF-QR cost about half an order of magnitude more than RBF-Direct in 1D

(b) RBF-QR costs multiple orders of magnitude more than RBF-Direct in 2D

Figure 4.6: For ε only as small as .1, the cost of RBF-QR becomes unsustainable in higher dimensions

The dominant cost of RBF-QR is the QR factorization of matrix Φ, which is of greater cost than the LU factorization used to solve the RBF-Direct system, although on the same order when $M \approx N$. Because of the aforementioned explosion

N	M
640	645
1280	1285
2560	2565
5120	5125

(a) Eigenfunction correction size remains reasonable in 1D

N	M
25	435
64	2346
121	7875
196	20100
289	43071
400	81810
529	142311

(b) The cost of storing the eigenfunction correction becomes unreasonable in 2D

Table 4.1: There is a significant increase between the maximum necessary series length from 1D to 2D.

in M as the dimension increases, there is no possible viability of RBF-QR for high dimensional problems. To combat that, this section explores the concept that M could be chosen smaller that N.

4.6.1 Low-rank approximation

Our goal for this section is to produce a low-rank approximation to the N-term RBF interpolant using $M < N$ eigenfunctions. The motivation behind this is to eliminate high-order eigenfunctions which contribute minimally to the solution, but greatly to the computational cost. Additionally, this may reduce the sensitivity of the solution to α as seen in Figure 4.3. The discussion from Section 4.5.2 shows that the choice of an "optimal" α depends on ε and is more sensitive with increasing M. We therefore hope that reducing M helps to mitigate the sensitivity.

To introduce this problem in the same context as Section 4.4, $M \leq N$ is fixed and all the eigenvalues λ_m with $M < m \leq N$ are set to zero. This results in an

approximate kernel matrix decomposition

$$\begin{aligned}\mathsf{K} &\approx \boldsymbol{\Phi}\tilde{\boldsymbol{\Lambda}}\boldsymbol{\Phi}^T \\ &= \begin{pmatrix} & & \\ \boldsymbol{\Phi}_1 & & \boldsymbol{\Phi}_2 \\ & & \end{pmatrix} \begin{pmatrix} \boldsymbol{\Lambda}_1 & \\ & 0 \end{pmatrix} \begin{pmatrix} & & \\ \boldsymbol{\Phi}_1 & & \boldsymbol{\Phi}_2 \\ & & \end{pmatrix}^T,\end{aligned}$$

where $\boldsymbol{\Phi}_1$ contains the first M eigenfunctions, $\boldsymbol{\Lambda}_1$ contains the first M (and only nonzero) eigenvalues, and $\boldsymbol{\Phi}_2$ contains the remaining $N - M$ eigenfunctions. Note that all these matrices are $N \times N$, and thus the QR decomposition from before is no longer necessary because $\boldsymbol{\Phi}^T$ is invertible.

Defining the basis transformation matrix X analogously to (4.10) we now have

$$\mathsf{X} = \tilde{\boldsymbol{\Lambda}}\boldsymbol{\Phi}^T,$$

and because $\tilde{\boldsymbol{\Lambda}}$ is not invertible we must instead consider the pseudoinverse [84]

$$\begin{aligned}\mathsf{X}^+ &= \boldsymbol{\Phi}^{-T}\tilde{\boldsymbol{\Lambda}}^+ \\ &= \boldsymbol{\Phi}^{-T} \begin{pmatrix} \boldsymbol{\Lambda}_1^{-1} & \\ & 0 \end{pmatrix}.\end{aligned}$$

This means that our new basis functions are

$$\Psi(\boldsymbol{x})^T = \boldsymbol{k}(\boldsymbol{x})^T\mathsf{X}^+,$$

which when expressed in terms of the eigenfunctions by expanding the kernel as

in (4.13) yields

$$\begin{aligned}
\Psi(\boldsymbol{x}) &= (\varphi_1(\boldsymbol{x}) \quad \dots \quad \varphi_N(\boldsymbol{x}))\Lambda\Phi^T\mathsf{X}^+ \\
&= (\varphi_1(\boldsymbol{x}) \quad \dots \quad \varphi_N(\boldsymbol{x}))\Lambda\Phi^T\Phi^{-T}\tilde{\Lambda}^+ \\
&= (\varphi_1(\boldsymbol{x}) \quad \dots \quad \varphi_N(\boldsymbol{x}))\Lambda\tilde{\Lambda}^+ \\
&= (\varphi_1(\boldsymbol{x}) \quad \dots \quad \varphi_N(\boldsymbol{x}))\begin{pmatrix} \mathsf{I}_M & \\ & 0 \end{pmatrix} \\
&= (\varphi_1(\boldsymbol{x}) \quad \dots \quad \varphi_M(\boldsymbol{x}) \quad 0 \quad \dots \quad 0)
\end{aligned}$$

analytically setting the last $N-M$ eigenfunctions equal to 0. Recasting the original linear system in this new basis then gives

$$\begin{aligned}
\Psi\boldsymbol{b} &= \boldsymbol{y} \\
\iff \quad \Phi\Lambda\Phi^T\mathsf{X}^+\boldsymbol{b} &= \boldsymbol{y} \\
\iff \quad \begin{pmatrix} & & \\ \Phi_1 & & \Phi_2 \\ & & \end{pmatrix} \begin{pmatrix} \mathsf{I}_M & \\ & 0 \end{pmatrix} \boldsymbol{b} &= \boldsymbol{y}.
\end{aligned}$$

As this is written, it is clearly a low rank system, which is appropriate since M nonzero functions are being fit to $N > M$ data points. There are two ways, identical in exact arithmetic, to solve this low-rank linear system in a least squares sense: the first uses the theoretically guaranteed invertibility of Φ in applying the pseudoinverse, i.e.,

$$\boldsymbol{b} = \begin{pmatrix} \mathsf{I}_M & \\ & 0 \end{pmatrix} \Phi^{-1}\boldsymbol{y}.$$

This of course requires forming $\Phi^{-1}\boldsymbol{y}$, which would subject this problem to the same sensitivity issues as before that stem from the unreasonably large M values given increasing N and/or ε. Moreover, inverting the matrix Φ would be more

costly than is necessary since the final solution is a least-squares solution of rank at most M whereas Φ has rank N. Instead, it seems that the more efficient and logical method of solving this system is to perform the matrix-matrix multiplication $\Phi\Lambda\Lambda^{+}$ analytically, leaving the system

$$\begin{pmatrix} \Phi_1 & 0 \end{pmatrix} \boldsymbol{b} = \boldsymbol{y}. \tag{4.21}$$

Zeroing out the eigenvalues analytically has the effect of ignoring the final $N - M$ components of the coefficient vector $\boldsymbol{b}$ during the solve, as should be expected. Solving this in a least-squares sense requires solving

$$\min_{\boldsymbol{b}} \left\| \begin{pmatrix} \Phi_1 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix} - \boldsymbol{y} \right\|_2^2 \quad \Longleftrightarrow \quad \min_{\boldsymbol{b}} \left\| \Phi_1 \boldsymbol{b}_1 - \boldsymbol{y} \right\|_2^2,$$

where the components $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ of the coefficient vector $\boldsymbol{b}$ are of size M and $N - M$, respectively. Following this logic, the solution is

$$\boldsymbol{b}_1 = \Phi_1^{+} \boldsymbol{y},$$

and $\boldsymbol{b}_2$ is unconstrained because the eigenfunctions associated with $\boldsymbol{b}_2$ are all identically zero.

4.6.2 Implementing truncation

The implementation of this regression approach is more straightforward than that of the interpolation problem because this system can be rephrased as an overdetermined least squares problem. One aspect that has thus far been omitted from our discussion is the selection of an M-value appropriate for early truncation. This choice is significant in reducing the computational complexity of the

approximation, but the most important factor in choosing M is producing a quality approximation.

To give an initial idea of the effect of M on the quality of the approximation, let us consider a simple scattered data fitting problem. Given $f(x) = \cos x + \mathrm{e}^{-(x-1)^2} + \mathrm{e}^{-(x+1)^2}$ and fixing $\varepsilon = \alpha = 1$, we consider N evenly spaced values of f on the interval $[-3, 3]$. Different values of N ranging from 10 to 500 are chosen to conduct the regression with five different sets of M-values corresponding to $\{.1N, .2N, .3N, .4N, .5N\}$. The approximation error curves are displayed in Figure 4.7.

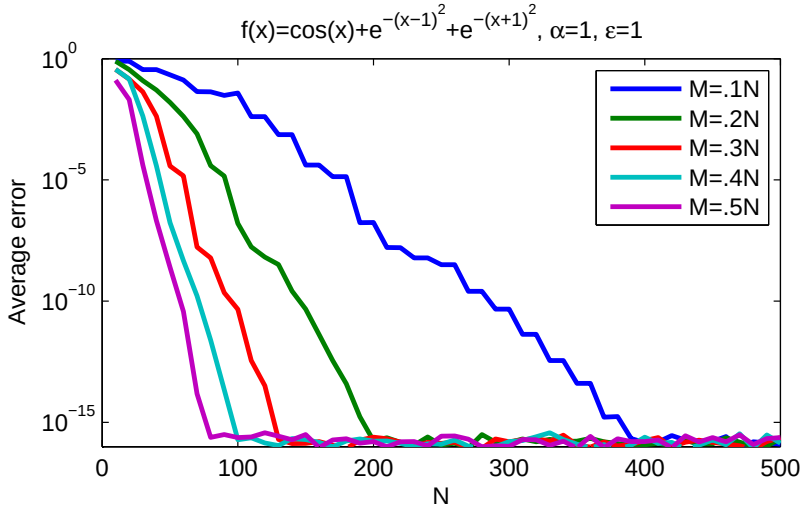


Figure 4.7: For any number N of data sites (and fixed $\varepsilon = \alpha = 1$), $M \approx 40$ eigenfunctions are adequate for optimal accuracy of the QR regression algorithm.

Regardless of the size of N, Figure 4.7 shows that the optimal accuracy of the approximation consistently occurs for the same value of $M \approx 40$. This is encouraging because it indicates possibly that for fixed ε, given any problem size N, there is a maximum space $\mathcal{R}$ that the eigenfunctions can effectively span, and that increasing M beyond $\mathcal{R}$ is not helpful.

The fact that the effective dimension of the space needed for an accurate kernel

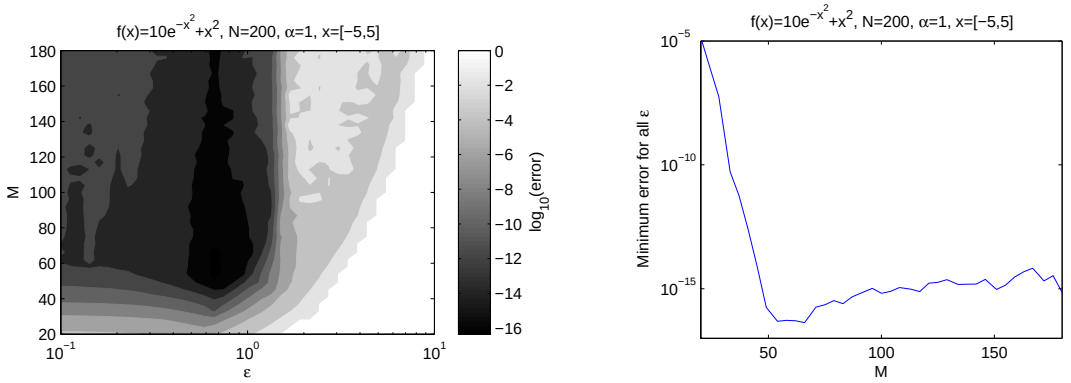
approximation is often rather small seems to be *mathematical folklore* and appears in different guises in various communities dealing with kernels. In the RBF literature we have, e.g., [75, 152] or the unpublished lecture notes [155]. In [155], for example, one can read that "there are low-rank subsystems which already provide good approximate solutions". In the statistical learning literature, the authors of [184] state that their main goal is "to find a fixed optimal m-dimensional linear model independent of the input data $\boldsymbol{x}$ [of sample size n], where the dimension m is also independent of n". Matrices that are of the same type as the kernel interpolation matrix K also arise in the method of fundamental solutions (MFS) for which the authors of [39] make similar observations regarding the non-trivial singular values, i.e., numerical rank, of K. In particular, in one of their examples [39, Figure 5] they see "no significant difference in accuracy using more than 40 [of 100] singular values".

To consider cases with varying ε, refer to Figure 4.8a. The contour plot there shows the approximation error, for fixed $N = 200$, obtained with values of M and ε that vary independently. The function used to generate the data for the approximation problem is

$$f_3(x) = 10\mathrm{e}^{-x^2} + x^2.$$

f_3 is a useful function because in the absence of the exponential function term, the polynomial alone would be best interpolated with M on the same order as the polynomial and $\varepsilon \to 0$ [60]. The additional term gives rise to a region of optimal M and ε centered around $(M, \varepsilon) \approx (60, 0.7)$ far from the $\varepsilon = 0$ axis. Also note that the M with least error is far away from the RBF-QR realm of $M > N$, although the difference in accuracy between the optimal error at $M = 66$ overall and the optimal error for $M = 180$ is small at $10^{-16.4}$ and $10^{-15.1}$ respectively. Here the error is computed with (4.20) by evaluating the interpolant at 1000 points in

$[-5, 5]$.



(a) M and ε both play key roles for accuracy.

(b) Beyond $M \approx 70$ the optimal error increases.

Figure 4.8: Over a range of ε values (with fixed N), experiments show an optimal M range.

Finding an optimal value of M is still an open problem, as it probably depends not only on the choice of ε, but also on such factors as the location of the data points, anisotropy in higher dimensions, the choice of α for (4.4b) and more. It is possible that future work can examine optimal values of both M and ε simultaneously to determine an optimal approximation. For the purposes of this work, we are interested primarily in exploring the $\varepsilon \to 0$ limit and thus we assume for future experiments that a good value of M is already chosen.

4.6.3 The effects of M and α on regression condition

Thus far we have been concerned exclusively with exploring the $\varepsilon \to 0$ limit for radial basis interpolation, which cannot be explored via the RBF-Direct approach because of ill-conditioning. Transitioning the traditional RBF problem into the RBF-QR formulation shifts the ill-conditioning from the radial basis functions to the eigenvalues which are inverted analytically. In Figure 4.3 it was shown that

the eigenfunctions could themselves become ill-conditioned for some values of α, and the sensitivity to the choice of α increased as the number of data points included in the problem, and thus the degree of the eigenfunctions, grew. This helped motivate RBF-QRr (QR regression) as discussed in Section 4.6.1, but in truncating the problem new issues arise.

Those issues can be isolated in the choices of series truncation value M and α, which first appeared in (4.2) when defining the weight function ρ. For RBF-QR as discussed in Section 4.4.2, α was chosen to represent the global size of the problem so that orthogonality via the weight function was best preserved, and M was chosen via (4.18) large enough so that $\lambda_M < \epsilon_{\text{mach}}\lambda_N$. In RBF-QRr we want to choose M at some value smaller than N, preferably much smaller for computational purposes.

It is known that for any fixed value of α, the truncated eigenfunction expansion provides the best M-term approximation (see, e.g., [167]). However, the precise relationship between α and M (and also ε) and condition is as yet unknown because we are no longer considering the entire space spanned by the Gaussians, but rather an optimal M-term approximation to it parametrized by α.

Examine Figure 4.9 as a snapshot of the typical condition of the least-squares regression system (4.21) for various values of M, α, N and ε. Here the condition number is defined as σ_1/σ_M where σ_k is the k^{th} largest singular value of the matrix Φ_1 as used in (4.21).

There are many implications to consider from Figure 4.9, keeping in mind that the purpose of RBF-QRr is to allow us to explore the $\varepsilon \to 0$ regime for RBF interpolation.

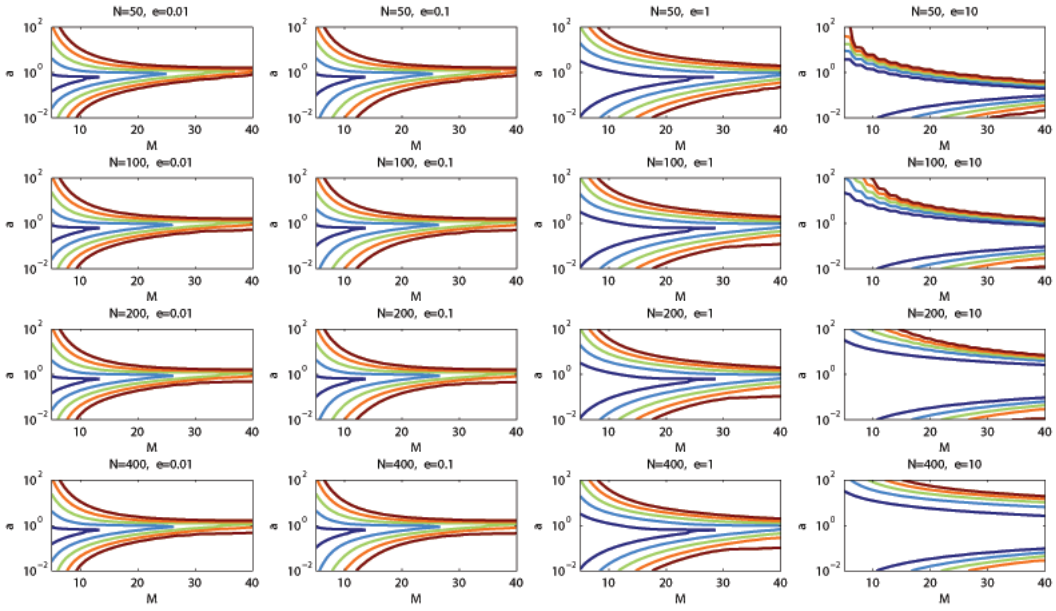


Figure 4.9: Contour lines at condition values of $\{10^2, 10^5, 10^8, 10^{11}, 10^{14}\}$. Data points are evenly spaced in $[-5, 5]$.

- As $\varepsilon \to 0$ (i.e., looking at the columns of plots from right to left) there seems to be a limiting condition distribution. This is to be expected as it has already been shown in, e.g., [115] and mentioned in Sections 4.3 and 4.5.2 that RBF interpolation reaches a polynomial limit as $\varepsilon \to 0$.
- As $\varepsilon \to \infty$ (corresponding to the right-most column of plots) greater values of M can be chosen without incurring a condition penalty. This corresponds to the RBF-Direct case where allowing $\varepsilon \to \infty$ produces more peaked/localized functions and a well-conditioned interpolation matrix.
- For a given M, there is a single value of α which produces the optimally conditioned system. When interpreted as the scale of the weight function ρ in (4.2) defining the inner product in which we measure orthogonality, it seems logical that there is a minimum α which best represents the scale. This was alluded to graphically in Figure 4.4.
- Increasing the number N of data points (i.e., looking at the plots from top

to bottom) has no negative effect on the condition. The only visible effect is with larger ε, which generates more localized Gaussians and in turn a better conditioned system. The $\varepsilon = 10$ pictures show the region of low condition significantly larger as N grows which we attribute to there being more "space" to fill with more eigenfunctions, and thus higher M is permitted without compromising condition.

- There is a region $M \in (M_{\max}, \infty)$ for which no α exists such that the condition of the system is less than $1/\epsilon_{\text{mach}}$. In some sense, this means that the dimension of the space we are interested in approximating can only be stably represented using $M < M_{\max}$ eigenfunctions. This reinforces our findings of Section 4.5.2.

Following from these ideas, the following guidelines for choosing M and α may be appropriate:

1. Given a value of ε, M should be chosen as large as possible under the constraint that the condition κ of the system (4.21) be less than $\kappa_{\max}$. $\kappa_{\max}$ would have to be chosen based on the specific problem.
2. α should be chosen to minimize the condition of the regression system.

Much of the motivation behind RBF-QRr was avoiding the computational cost of RBF-QR by solving smaller systems. At present we see no way to satisfy those guidelines without significantly increasing the cost of RBF-QRr; it should be noted though that the cost is still less than RBF-QR. These guidelines are meant to be a summary of our insights thus far, and hopefully a starting point for future research.

For the following experiments, multiple M values are sampled and one is chosen to represent RBF-QRr; when experiments have RBF-QR results, those M values

are be chosen via (4.17). For each ε, α is chosen as small as possible such that orthogonality is guaranteed for the first 25 eigenfunctions. Keep in mind that these are just choices made to display the usefulness of RBF-QRr to explore the $\varepsilon \to 0$ region, and more work needs to be done to determine optimal M and α values in general.

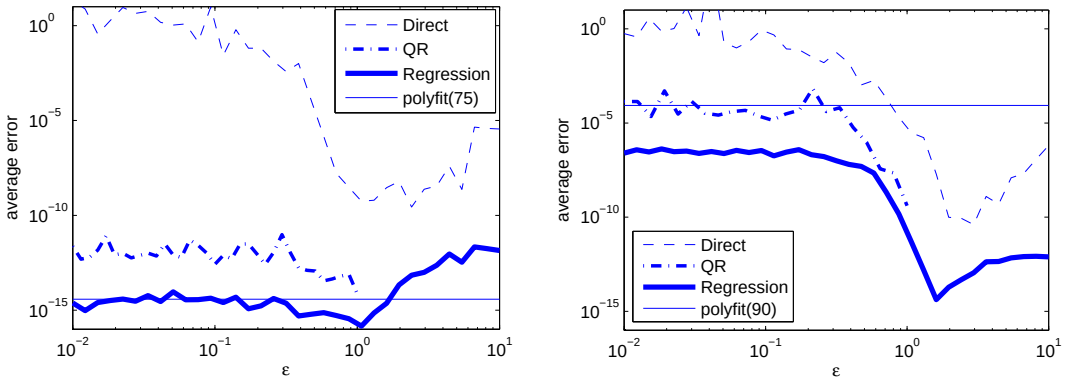
4.7 Numerical experiments for regression

In this section we provide comparisons of the RBF-QRr regression algorithm to the RBF-direct method and — in Section 4.7.1 to RBF-QR as described in Section 4.4 — for various data sets in various space dimensions. In each of these experiments, the truncation range for the value of M used in the regression is specified.

4.7.1 1D approximation

In this series of experiments the data is generated by two different univariate functions f evaluated at N evenly spaced points.

For Figure 4.10a we reprise the function from Figure 4.7 on the domain $x \in [-3, 3]$ and see that better accuracy can be achieved with RBF-QRr *approximation* instead of RBF-QR *interpolation*, even at less cost. In Figure 4.10b RBF-QRr maintains higher accuracy than both RBF-QR and RBF-Direct. Note that the function used in Figure 4.10b is the notorious Runge function on the domain $x \in [-4, 4]$ and is much harder to approximate by polynomials than the function used in Figure 4.10a. In fact, we can see that the MATLAB algorithm `polyfit` is no longer stable and the "flat limit" Gaussian approximation, which uses orthogonal



(a) $f(x) = \cos x + \mathrm{e}^{-(x-1)^2} - \mathrm{e}^{-(x+1)^2}$, $N = 150$, $M = 75$

(b) $f(x) = (1+x^2)^{-1}$, $N = 150$, $M = 90$

Figure 4.10: Regression avoids both the sensitivity in RBF-QR associated with large N, and the $\varepsilon \to 0$ ill-conditioning in RBF-Direct.

polynomials instead of a Vandermonde matrix, is considerably more accurate than the polynomial interpolant. Moreover, the Gaussian approximation for $\varepsilon \approx 1.5$ is many orders of magnitude more accurate that the polynomial interpolant.

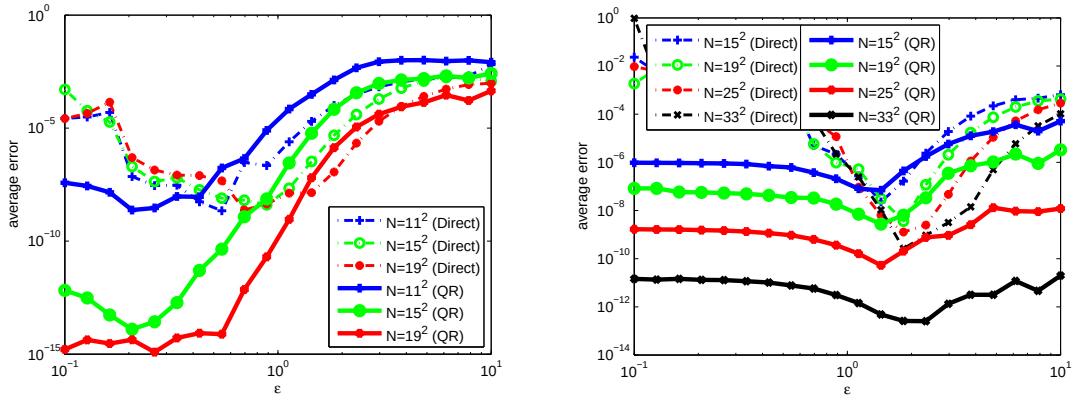
What should be noted in these two graphs is that RBF-QRr fails to reproduce RBF-Direct as ε grows and RBF-Direct is sufficiently well conditioned. This is because the eigenvalues decay more slowly for larger ε and thus the choice of M small is no longer appropriate. $M > N$ would be required to conduct the approximation as $\varepsilon \to \infty$, but there would be no reason to use RBF-QR in that realm because RBF-Direct is well conditioned.

4.7.2 Higher-dimensional approximation

One of the great benefits of considering radial basis functions for interpolation is their natural adaptation to use in higher dimensions. That flexibility is not lost when using an eigenfunction expansion to approximate the Gaussian, as was ini-

tially described in Section 4.3.2. For the experiments in this section only regression is considered because of the computational cost of RBF-QR in higher dimensions. α is chosen to satisfy orthogonality for up to the fourth eigenfunction; this choice is somewhat arbitrary, but seems to produce good results in the $\varepsilon \to 0$ region of interest.

In Figure 4.11 we present examples of RBF-QRr compared to RBF-Direct for two different test functions. The data is generated by sampling these functions at N evenly spaced points (see, e.g., [60]) in the region $[-1, 1]^2$. As before, RBF-QRr drifts further from the true RBF interpolant for large values of the shape parameter ε because the necessary number of eigenfunctions to conduct a quality approximation is too great to complete a regression. For this region, it is not necessary to use RBF-QRr because RBF-Direct has acceptable condition, but it is worth noting that also the RBF-QRr method has its limitations.

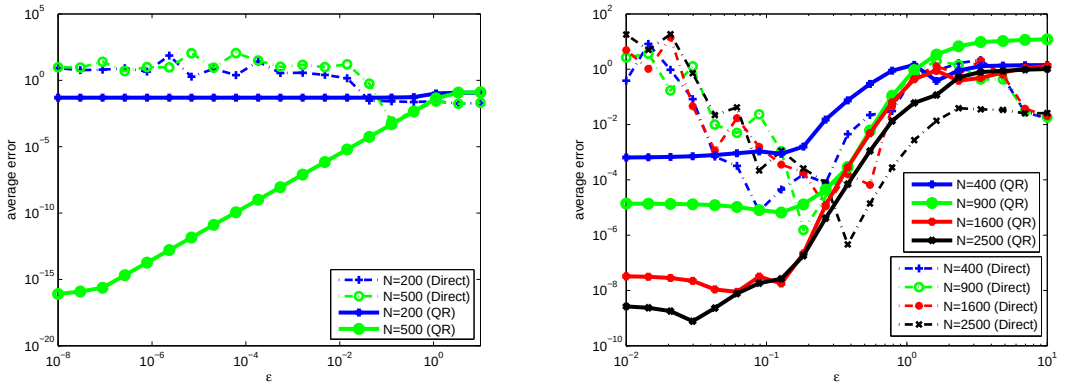


(a) $f(x, y) = \cos(x + y)$, $M = .5N$ (b) $f(x, y) = (x^2 + y^2)^{-1}$, $M = .7N$

Figure 4.11: Comparison of RBF-Direct and RBF-QRr regression in 2D using various evenly spaced data points in $[-1, 1]^2$.

Just as in the 2D setting, eigenfunction expansions work for higher-dimensional settings as well. Figure 4.12 shows examples for two functions of five variables with very different ε profiles. As one would expect, the polynomial in Figure 4.12a is

reproduced to within machine precision as soon as enough eigenfunctions are used and ε is chosen small enough (note that the dimension of the space of polynomials of degree five in five variables is 252). For the trigonometric test function illustrated in Figure 4.12b the RBF-QRr method is again more accurate and more stable than RBF-Direct. However, the accuracy of the approximation is weak for larger ε, as was seen previously.



(a) $f(\boldsymbol{x}) = 1+(x_1+x_2+x_3)^2(x_4-x_5)^2(x_1+x_4)$ (b) $f(\boldsymbol{x}) = \cos(.2(x_1+x_2+x_3+x_4+x_5))$

Figure 4.12: Comparison of RBF-Direct and RBF-QRr regression in 5D using a different number, N, of Halton data points in $[-1,1]^5$.

4.8 Conclusions and remarks about future work

The stated purpose of this chapter was to provide a technique to allow for stable evaluation of RBF interpolants when the shape parameter values are so small that ill-conditioning overwhelms the traditional approach to RBF interpolation. This "flat-limit" regime is of particular practical interest since this often corresponds to the range of the shape parameter that provides the most accurate RBF interpolant (provided it can be stably computed and evaluated). By adding this accurate interpolant to our infrastructure, we can couple multiphysics simulations via stable kernel-based interpolation. This is described in Chapter 6.

Our initial approach closely followed [72] replacing the use of spherical harmonics with an eigenfunction decomposition parametrized by a value α related to the global scale of the problem. This technique consists of replacing the Gaussian basis functions centered at the N data sites with $M > N$ new basis functions that reproduce the Gaussian kernel within the limits of machine precision. The choice of this new basis was driven by the desire to have condition properties superior to the Gaussian for sufficiently small ε, but it introduces some redundancy in the representation of the N-dimensional Gaussian approximation space which leads to some of the conditioning problems observed in Figure 4.3.

For certain values of N and ε this approach worked well, but for larger values of N we encountered limitations incurred by the condition of the eigenfunctions which were absent from the work involving spherical harmonics. To compensate for this new source of ill-conditioning (independent of the shape parameter) a new approach was devised involving $M < N$ basis functions and a least squares solution to the approximation problem. This technique overcame the ill-conditioning of the interpolation problem using careful choices of α and M to balance the condition of the problem against producing the best approximation to the space spanned by the Gaussians.

Given that we have seen the potential for success with this eigenfunction approximation of the Gaussian kernels, there is still much to be investigated to fully understand the work started here. We end by briefly discussing some of these topics, and pointing to sections where those problems are addressed. These issues vary in complexity and significance, but their thorough understanding would require much more research than can be included here.

4.8.1 Location of data points

In our work thus far, we have studied input data on an evenly spaced grid, on the Chebyshev points and at the Halton points. Overall we have noticed very little effect of the data point distribution on the condition and accuracy of the RBF-QR and RBF-QRr solutions provided the domain is "covered well" by the data points. Is this true in general; will the distribution of points have little or no significance on the effectiveness of RBF-QR and RBF-QRr?

4.8.2 Analytic relationship of the parameters ε, M and α

In Figure 4.9 we illustrated how the truncation value M and global scale parameter α affect the condition of the RBF-QRr regression algorithm for given values of ε and varying problem size N. Rigorous analysis needs to be done on the relationship between α and ε. Every Gaussian kernel with shape parameter ε has a family of equivalent eigenfunction expansions parametrized by α. In exact arithmetic with $M \to \infty$ all of these series are equal to the Gaussian kernel, but for a finite M there are significant differences which may lead to ill-conditioned systems. Can we determine analytically what α-value is appropriate for each M, or if there even always exists such a (unique) value?

4.8.3 Anisotropic approximation

In this chapter we have worked under the assumption that the same shape parameter ε should be used in each space dimension. As mentioned in Section 4.3.2, the theoretical possibilities of our eigenfunction-based QR algorithms are much more

general. We could choose to write the kernel as $K(\boldsymbol{x}, \boldsymbol{z}) = \exp((\boldsymbol{x} - \boldsymbol{z})^T \mathsf{E}(\boldsymbol{x} - \boldsymbol{z}))$, where the standard isotropic Gaussian would correspond to $\mathsf{E} = \varepsilon^2 \mathsf{I}_N$. The derivation in Section 4.3.2 provides a natural route to using eigenfunction expansions for anisotropic Gaussians (i.e., with E diagonal, but not a scalar multiple of the identity). However, in so doing there is also the opportunity to use a strategy to employ different choices of M and α in different dimensions. What flexibility and accuracy does this added freedom offer? How does this affect the complexity of the implementation and execution of the method? A different choice of E (still positive definite) would result in a different kernel and more flexibility when conducting interpolation in higher dimensions. Can the theoretical foundation be extended to cover eigenfunction decompositions for a nondiagonal E?

4.8.4 Other kernels

We have thus far only considered the Gaussian kernel and its associated eigenexpansion. There are many other positive definite kernels (see, e.g., [60]) that involve a shape parameter for which the RBF-Direct method is associated with the trade-off principle, i.e., increased accuracy comes at the price of a loss in numerical stability. In [70] some ideas for the oscillatory Bessel or Poisson kernels are presented. What about inverse multiquadrics, Matérn kernels, and many others?

CHAPTER 5

COMPUTATIONAL IMPROVEMENTS FOR EIGENFUNCTIONS

5.1 Introduction

The 1D Gaussian,

$$K(x,z) = \mathrm{e}^{-\varepsilon^2(x-z)^2},$$

is a common choice of radial basis function (RBF) because of its attractive approximation properties (see, e.g., [61, 176]). Applications ranging from statistics, machine learning, computer graphics, and boundary value problems have all found Gaussians useful, and at times optimal, for sufficiently smooth problems. Because Gaussians approach polynomials in the $\varepsilon \to 0$ limit [52, 115], techniques using polynomials may potentially benefit from a kernel-based approach.

One of the significant barriers to realizing this potential benefit is the ill-conditioning associated with the $\varepsilon \to 0$ limit, as discussed in [60, 151]. In the previous chapter, an eigenfunction expansion was developed to produce a stable basis for Gaussian RBFs. For small ε the series converges very quickly, allowing for a compact approximation to the Gaussians of interest without the ill-conditioning.

As discussed in Chapter 4, the eigenfunction expansion has the form

$$K(x,z) = \sum_{k=1}^{M} \lambda_k \varphi_k(x)\varphi_k(z), \tag{5.1}$$

where M is the truncation point of the otherwise infinite series, and

$$\lambda_k = \sqrt{\frac{\alpha^2}{\alpha^2+\delta^2+\varepsilon^2}}\left(\frac{\varepsilon^2}{\alpha^2+\delta^2+\varepsilon^2}\right)^{k-1},$$
$$\varphi_k(x) = \gamma_k \mathrm{e}^{-\delta^2 x^2} H_{k-1}(\beta\alpha x).$$

The value α is the global scale parameter as defined in Section 4.3, and the auxiliary parameters

$$\beta = \left(1 + \left(\frac{2\varepsilon}{\alpha}\right)^2\right)^{\frac{1}{4}}, \quad \gamma_k = \sqrt{\frac{\beta}{2^{k-1}\Gamma(k)}}, \quad \delta^2 = \frac{\alpha^2}{2}\left(\beta^2 - 1\right),$$

are fixed once α and ε are chosen. Working with these eigenfunctions allows for stable computations with Gaussians, but can be more costly than working in the traditional basis because of the need to perform a QR factorization.

Because these eigenfunctions involve Hermite polynomials, they are also defined by a three-term recurrence. That recurrence relation is

$$\begin{aligned}
\varphi_1(x) &= \sqrt{\beta}\mathrm{e}^{-\delta^2 x^2}, \\
\varphi_2(x) &= \sqrt{2}\beta\alpha x\sqrt{\beta}\mathrm{e}^{-\delta^2 x^2}, \\
\varphi_{k+1}(x) &= \sqrt{\frac{2}{k}}\beta\alpha x\varphi_k(x) - \sqrt{\frac{k-1}{k}}\varphi_{k-1}(x), \qquad k \geq 1,
\end{aligned} \tag{5.2}$$

and is only valid in one dimension; multiple dimensions rely on the one dimensional case, and will potentially be considered later.

This chapter derives a fast QR decomposition of the matrix $(\Phi)_{n,m} = \varphi_m(x_n)$ using the recurrence relation above. The matrix Φ appears in scattered data approximation using Gaussians: the traditional Gaussian interpolant is defined by the $N \times N$ system

$$\mathsf{K}\boldsymbol{c} = \boldsymbol{y}, \tag{4.14}$$

where $(\mathsf{K})_{i,j} = K(x_i, x_j)$. Here $\boldsymbol{x} = (x_1 \ \cdots \ x_N)^T$ are the function value locations, $\boldsymbol{y}$ are the function values at $\boldsymbol{x}$ and $\boldsymbol{c}$ is the coefficient vector, so that the interpolant s can be evaluated via the inner product

$$s(x) = \left(K(x, x_1) \ \cdots \ K(x, x_N)\right)\boldsymbol{c}.$$

The eigenfunction expansion (5.1) with $M < N$ yields the approximation $\mathsf{K} \approx \mathsf{\Phi\Lambda\Phi}^T$. By making this substitution into (4.14), we are left with the low rank system

$$\mathsf{\Phi\Lambda\Phi}^T \boldsymbol{c} = \boldsymbol{y},$$

or the least squares system

$$\mathsf{\Phi}\tilde{\boldsymbol{c}} = \boldsymbol{y}, \tag{5.3}$$

as described in Section 4.6.1, where $\tilde{\boldsymbol{c}} = \mathsf{\Lambda\Phi}^T \boldsymbol{c}$. This derivation was discussed in Section 4.6.

The system (5.3) benefits from the fast QR decomposition

$$\mathsf{\Phi} = \mathsf{Q\Sigma U} \tag{5.4}$$

derived in this chapter. The recurrence relation for Hermite polynomials, described in Section 5.1.1, is extended to Gaussian eigenfunctions in Section 5.1.2. Two algorithms are presented to produce (5.4): one based on computing U^{-1} in Section 5.3 and the other based on computing Q in Section 5.4. These sections are based in part on

M. McCourt, *A fast least squares solver for stable Gaussian computations*, Proceedings of the Eighth International Conference on Scientific Computing and Applications, accepted

which is cited as [122]. The error properties of these algorithms are analyzed in Section 5.4.3 to determine their viability in solving systems. These algorithms are also studied as a potential method to determine an appropriate M for the Gaussian approximation. Finally, Section 5.6 introduces iterative linear solvers involving eigenfunctions, but does not draw any significant conclusions.

5.1.1 Recurrence relations for Hermite polynomials

Much of this section parallels the methods from [81], and serves to introduce relevant theory involving Hermite polynomials. That work dealt with orthogonal polynomials in general, but because of the structure of Gaussian eigenfunctions, we concern ourselves only with Hermite polynomials. Hermite polynomials satisfy the recurrence relation

$$\begin{aligned} H_0(\beta\alpha x) &= 1, \\ H_1(\beta\alpha x) &= 2\beta\alpha x, \\ H_k(\beta\alpha x) &= 2\beta\alpha x H_{k-1}(\beta\alpha x) - 2(k-1)H_{k-2}(\beta\alpha x), \qquad k \geq 2. \end{aligned} \tag{5.5}$$

Note the $\beta\alpha$ factor is not necessary, but rather is inserted here to preserve consistency with later sections. The matrices

$$\mathsf{H} = \begin{pmatrix} H_0(\beta\alpha x_1) & \cdots & H_{M-1}(\beta\alpha x_1) \\ & \vdots & \\ H_0(\beta\alpha x_N) & \cdots & H_{M-1}(\beta\alpha x_N) \end{pmatrix}, \qquad \mathsf{X} = \begin{pmatrix} 1 & \cdots & (\beta\alpha x_1)^{M-1} \\ & \vdots & \\ 1 & \cdots & (\beta\alpha x_N)^{M-1} \end{pmatrix}$$

satisfy the relationship $\mathsf{H} = \mathsf{X}\mathsf{L}^T$, where the lower triangular matrix $(\mathsf{L})_{k,j} = \ell_{k,j}$ is defined by the Hermite coefficients

$$H_{k-1}(\beta\alpha x) = \sum_{j=1}^{M} \ell_{k,j} (\beta\alpha x)^{j-1}.$$

These matrices have a special structure: X is a Vandermonde matrix [84], and H is a Vandermonde-like matrix [32]. Often Vandermonde matrices are undesirable, because despite their simplistic structure they are rather ill-conditioned. Vandermonde-like matrices, generated by orthogonal polynomials, often have much better condition [79].

When the points x_k are distinct ($x_k \neq x_j$ for $k \neq j$), $\mathsf{H}^T\mathsf{H}$ is symmetric positive

definite, and we know it has an LDL factorization

$$\begin{aligned}\mathsf{H}^T\mathsf{H} &= \mathsf{L}\mathsf{X}^T\mathsf{X}\mathsf{L}^T \\ &= \mathsf{U}^T\mathsf{D}\mathsf{U}.\end{aligned}$$

This is useful, but does not directly allow for a fast factorization. Rather, the Hankel matrix $\mathsf{X}^T\mathsf{X}$ does have a fast method for computing the factorization

$$\mathsf{X}^T\mathsf{X} = \tilde{\mathsf{U}}^T\tilde{\mathsf{D}}\tilde{\mathsf{U}}$$

using the recurrence relation

$$\tilde{\boldsymbol{u}}_{k+1} = (\mathsf{Z} + c_{k+1}\mathsf{I}_M)\tilde{\boldsymbol{u}}_k - b_k\tilde{\boldsymbol{u}}_{k-1}, \tag{5.6}$$

where $\tilde{\boldsymbol{u}}_k$ is the kth column of $\tilde{\mathsf{U}}^{-1}$. This was presented in [49]. Above, c_{k+1} and b_k are related to the location of the points $\boldsymbol{x}$, and

$$\mathsf{Z} = \begin{pmatrix} 0 & 0 \\ \mathsf{I}_{M-1} & 0 \end{pmatrix}.$$

For a symmetric distribution of $\boldsymbol{x}$ around 0, $c_{k+1} = 0$.

In [81], the author extends (5.6) to orthogonal polynomials which are constructed using a three-term recursion. Section 5.1.2 performs a similar derivation on the eigenfunctions.

5.1.2 Recurrence relations for Gaussian eigenfunctions

The structure of Φ from (5.3) is significant, because it can be factored as

$$\begin{pmatrix} \varphi_1(x_1) & \cdots & \varphi_M(x_1) \\ & \vdots & \\ \varphi_1(x_N) & \cdots & \varphi_M(x_N) \end{pmatrix} = \begin{pmatrix} e^{-\delta^2 x_1^2} & & \\ & \ddots & \\ & & e^{-\delta^2 x_N^2} \end{pmatrix} \begin{pmatrix} 1 & \cdots & (\beta\alpha x_1)^{M-1} \\ & \vdots & \\ 1 & \cdots & (\beta\alpha x_N)^{M-1} \end{pmatrix} \times \begin{pmatrix} \ell_{1,1} & \cdots & \ell_{M,1} \\ & \ddots & \vdots \\ & & \ell_{M,M} \end{pmatrix} \begin{pmatrix} \gamma_1 & & \\ & \ddots & \\ & & \gamma_M \end{pmatrix}$$

or, more succinctly, as

$$\Phi = \mathsf{DXL}^T\mathsf{G} = \mathsf{DHG}, \tag{5.7}$$

where D is diagonal with $e^{-\delta^2 x_n^2}$ in the nth position, G is diagonal with γ_m in the mth position, and X and L are defined in Section 5.1.1. We may also at times use the notation

$$\Phi = \begin{pmatrix} & & \\ \boldsymbol{\phi}_1(\boldsymbol{x}) & \cdots & \boldsymbol{\phi}_M(\boldsymbol{x}) \\ & & \end{pmatrix},$$

where

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}, \qquad \boldsymbol{\phi}_k(\boldsymbol{x}) = \begin{pmatrix} \varphi_k(x_1) \\ \vdots \\ \varphi_k(x_N) \end{pmatrix}.$$

Let us take this opportunity to prove the three-term recurrence (5.2). The early terms can be constructed by directly substituting in the Hermite polynomials of degree 0,

$$\varphi_1(x) = \underbrace{\gamma_1}_{\sqrt{\beta}} \exp(-\delta^2 x^2) \underbrace{H_0(\beta\alpha x)}_{=1} = \sqrt{\beta}\mathrm{e}^{-\delta^2 x^2},$$

and degree 1,

$$\varphi_2(x) = \underbrace{\gamma_2}_{\sqrt{\beta/2}} \exp(-\delta^2 x^2) \underbrace{H_1(\beta\alpha x)}_{=2\beta\alpha x} = \sqrt{2}\beta\alpha x\sqrt{\beta}\mathrm{e}^{-\delta^2 x^2}.$$

Starting from (5.5) we can multiply by $\gamma_{k+1}\mathrm{e}^{-\delta^2 x^2}$ to start the conversion to eigenfunctions:

$$\underbrace{\gamma_{k+1}\mathrm{e}^{-\delta^2 x^2} H_k(\beta\alpha x)}_{=\varphi_{k+1}(x)} = 2\beta\alpha x\gamma_{k+1}\mathrm{e}^{-\delta^2 x^2} H_{k-1}(\beta\alpha x) - 2(k-1)\gamma_{k+1}\mathrm{e}^{-\delta^2 x^2} H_{k-2}(\beta\alpha x).$$

Now we can examine the structure of γ_{k+1} to see

$$\gamma_{k+1} = \sqrt{\frac{\beta}{2^k\Gamma(k+1)}} = \sqrt{\frac{1}{2k}}\sqrt{\frac{\beta}{2^{k-1}\Gamma(k)}} = \sqrt{\frac{1}{2k}}\gamma_k,$$

$$\gamma_{k+1} = \sqrt{\frac{\beta}{2^k\Gamma(k+1)}} = \sqrt{\frac{1}{4k(k-1)}}\sqrt{\frac{\beta}{2^{k-2}\Gamma(k-1)}} = \sqrt{\frac{1}{4k(k-1)}}\gamma_{k-1}.$$

Plugging those in above gives

$$\varphi_{k+1}(x) = 2\beta\alpha x\sqrt{\frac{1}{2k}}\underbrace{\gamma_k\mathrm{e}^{-\delta^2 x^2} H_{k-1}(\beta\alpha x)}_{=\varphi_k(x)} - 2(k-1)\sqrt{\frac{1}{4k(k-1)}}\underbrace{\gamma_{k-1}\mathrm{e}^{-\delta^2 x^2} H_{k-2}(\beta\alpha x)}_{=\varphi_{k-1}(x)},$$

which, after simplifying the coefficients, produces the recurrence (5.2).

5.2 Developing a recurrence for the QR factors

In [81] the author develops a fast method to compute the Q, Σ and U components from (5.4) when the Φ matrix consists of terms from an orthogonal polynomial. This section uses an analogous derivation to factorize the Φ matrix when it is built from the 1D Gaussian eigenfunctions. We begin in Section 5.2.1 by exploiting the structure of the matrix $\Phi^T\Phi$ to construct U, and compute the rest of the decomposition in the remaining subsections.

5.2.1 A recurrence for the upper triangular factor

Following an analogous statement from earlier, both operators

$$\mathsf{X}^T\mathsf{D}^2\mathsf{X} = \tilde{\mathsf{U}}^T\tilde{\mathsf{\Sigma}}^2\tilde{\mathsf{U}}, \tag{5.8}$$

$$\mathsf{\Phi}^T\mathsf{\Phi} = \mathsf{U}^T\mathsf{\Sigma}^2\mathsf{U} \tag{5.9}$$

have LDL factorizations because they are positive definite. Unfortunately, computing (5.8) is dangerous because of the ill-conditioning associated with the Vandermonde matrix X. This is the same ill-conditioning mentioned earlier with the $\mathsf{X}^T\mathsf{X}$ matrix, although for the eigenfunctions, the relevant inner product is $(\mathsf{DX})^T(\mathsf{DX})$, which yields the additional factor D^2 seen above.

Though we do not want to work with X, $\mathsf{X}^T\mathsf{D}^2\mathsf{X}$ is a Hankel matrix (see [18] for a proof), and (5.6) does apply, which provides a foundation for computing U^{-1}. Using (5.7), (5.8), and (5.9) we can develop a direct relationship between the columns of U^{-1} and $\tilde{\mathsf{U}}^{-1}$:

$$\begin{aligned}
\mathsf{\Phi}^T\mathsf{\Phi} &= (\mathsf{DXL}^T\mathsf{G})^T\mathsf{DXL}^T\mathsf{G} \\
&= \mathsf{GL}(\mathsf{X}^T\mathsf{D}^2\mathsf{X})\mathsf{L}^T\mathsf{G}, \\
\mathsf{U}^T\mathsf{\Sigma}^2\mathsf{U} &= \mathsf{GL}(\tilde{\mathsf{U}}^T\tilde{\mathsf{\Sigma}}^2\tilde{\mathsf{U}})\mathsf{L}^T\mathsf{G}, \\
(\mathsf{\Sigma U})^T(\mathsf{\Sigma U}) &= (\tilde{\mathsf{\Sigma}}\tilde{\mathsf{U}}\mathsf{L}^T\mathsf{G})^T\tilde{\mathsf{\Sigma}}\tilde{\mathsf{U}}\mathsf{L}^T\mathsf{G}.
\end{aligned}$$

Because both $\mathsf{\Sigma U}$ and $\tilde{\mathsf{\Sigma}}\tilde{\mathsf{U}}\mathsf{L}^T\mathsf{G}$ are upper triangular, the relation

$$\begin{aligned}
\mathsf{\Sigma U} &= \tilde{\mathsf{\Sigma}}\tilde{\mathsf{U}}\mathsf{L}^T\mathsf{G}, \\
\tilde{\mathsf{\Sigma}}^{-1}\mathsf{\Sigma U} &= \tilde{\mathsf{U}}\mathsf{L}^T\mathsf{G} \\
\mathsf{U}^{-1}\left(\tilde{\mathsf{\Sigma}}^{-1}\mathsf{\Sigma}\right)^{-1} &= \left(\mathsf{L}^T\mathsf{G}\right)^{-1}\tilde{\mathsf{U}}^{-1}, \\
\left(\mathsf{L}^T\mathsf{G}\right)\mathsf{U}^{-1}\left(\tilde{\mathsf{\Sigma}}^{-1}\mathsf{\Sigma}\right)^{-1} &= \tilde{\mathsf{U}}^{-1}
\end{aligned}$$

must hold. Studying this at the column level produces the relationship

$$\frac{1}{s_k}\mathsf{L}^T\mathsf{G}\boldsymbol{u}_k = \tilde{\boldsymbol{u}}_k, \qquad 1 \le k \le M, \tag{5.10}$$

where s_k is the kth diagonal value of $\tilde{\Sigma}^{-1}\Sigma$ and $\boldsymbol{u}_k$ is the kth column of the matrix U^{-1}.

The product $(\mathsf{L}^T\mathsf{G})\tilde{\mathsf{U}}^{-1}$ is between upper triangular matrices $\mathsf{L}^T\mathsf{G}$ and $\tilde{\mathsf{U}}^{-1}$, thus the diagonal values of the product are the product of the diagonal elements. Because the diagonal values of $\tilde{\mathsf{U}}^{-1}$ are all 1 (by design), the diagonal of $(\mathsf{L}^T\mathsf{G})\tilde{\mathsf{U}}^{-1}$ is equal to the diagonal of $\mathsf{L}^T\mathsf{G}$. Therefore, since the diagonal values of U^{-1} are also 1, we know that the diagonal matrix $\tilde{\Sigma}^{-1}\Sigma$ has entries equal to the diagonal of $\mathsf{L}^T\mathsf{G}$. To ensure the unit diagonal of U^{-1}, we require

$$s_k = \ell_{k,k}\gamma_k = 2^{k-1}\sqrt{\frac{\beta}{2^{k-1}\Gamma(k)}} = \sqrt{\frac{2^{k-1}\beta}{\Gamma(k)}}\,. \tag{5.11}$$

Note that $\ell_{k,k} = 2^{k-1}$ appears on the diagonal of L as the coefficient of the leading term of the $k-1$ Hermite polynomial.

Making the replacement from (5.10) in (5.6) produces

$$\begin{aligned}\frac{1}{s_{k+1}}\mathsf{L}^T\mathsf{G}\boldsymbol{u}_{k+1} &= (\mathsf{Z} + c_{k+1}\mathsf{I}_M)\frac{1}{s_k}\mathsf{L}^T\mathsf{G}\boldsymbol{u}_k - b_k\frac{1}{s_{k-1}}\mathsf{L}^T\mathsf{G}\boldsymbol{u}_{k-1}\,,\\ \boldsymbol{u}_{k+1} &= (\mathsf{G}^{-1}\mathsf{L}^{-T}\mathsf{Z}\mathsf{L}^T\mathsf{G} + c_{k+1}\mathsf{I}_M)\frac{s_{k+1}}{s_k}\boldsymbol{u}_k - b_k\frac{s_{k+1}}{s_{k-1}}\boldsymbol{u}_{k-1}\,.\end{aligned}$$

Simplifying this using (5.11), and fixing $\boldsymbol{u}_{-1} = 0$, produces the recurrence

$$\boldsymbol{u}_{k+1} = (\mathsf{G}^{-1}\mathsf{L}^{-T}\mathsf{Z}\mathsf{L}^T\mathsf{G} + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\boldsymbol{u}_k - \frac{2b_k}{\sqrt{k^2-k}}\boldsymbol{u}_{k-1}, \quad 1 \le k \le M. \tag{5.12}$$

To reduce the cost of computing this, we need to simplify $\mathsf{G}^{-1}\mathsf{L}^{-T}\mathsf{Z}\mathsf{L}^T\mathsf{G}$. To do

this, we need to establish the relationship

$$x\begin{pmatrix}\varphi_1(x)\\ \varphi_2(x)\\ \vdots\\ \varphi_M(x)\end{pmatrix} = \mathsf{T}\begin{pmatrix}\varphi_1(x)\\ \varphi_2(x)\\ \vdots\\ \varphi_M(x)\end{pmatrix} + \begin{pmatrix}0\\ 0\\ \vdots\\ \varphi_{M+1}(x)\frac{1}{\beta\alpha}\sqrt{\frac{M}{2}}\end{pmatrix}, \tag{5.13}$$

where T is the symmetric tridiagonal matrix

$$\mathsf{T} = \frac{1}{\sqrt{2}\beta\alpha}\begin{pmatrix}0 & 1 & & & \\ 1 & 0 & \sqrt{2} & & \\ & \sqrt{2} & 0 & \ddots & \\ & & \ddots & \ddots & \sqrt{M-1}\\ & & & \sqrt{M-1} & 0\end{pmatrix}.$$

The rows of the system (5.13) comes directly from the three-term recurrence (5.2), which can be written as

$$x\varphi_k(x) = \sqrt{\frac{k-1}{2}}\frac{1}{\beta\alpha}\varphi_{k-1}(x) + \sqrt{\frac{k}{2}}\frac{1}{\beta\alpha}\varphi_{k+1}(x).$$

Note that given the structure of the eigenfunctions (5.7), we know $\mathsf{\Phi}^T = \mathsf{GLX}^T\mathsf{D}$, and at a column level we can write

$$\begin{pmatrix}\varphi_1(x)\\ \varphi_2(x)\\ \vdots\\ \varphi_M(x)\end{pmatrix} = \begin{pmatrix}\gamma_1 e^{-\delta^2x^2}H_0(\beta\alpha x)\\ \gamma_2 e^{-\delta^2x^2}H_1(\beta\alpha x)\\ \vdots\\ \gamma_M e^{-\delta^2x^2}H_{M-1}(\beta\alpha x)\end{pmatrix} = e^{-\delta^2x^2}\mathsf{GL}\begin{pmatrix}1\\ \beta\alpha x\\ \vdots\\ (\beta\alpha x)^{M-1}\end{pmatrix}.$$

Applying $\mathsf{L}^{-1}\mathsf{G}^{-1}$ to (5.13) produces the equation

$$xe^{-\delta^2x^2}\begin{pmatrix}1\\ \beta\alpha x\\ \vdots\\ (\beta\alpha x)^{M-1}\end{pmatrix} = \mathsf{L}^{-1}\mathsf{G}^{-1}\mathsf{TGL}e^{-\delta^2x^2}\begin{pmatrix}1\\ \beta\alpha x\\ \vdots\\ (\beta\alpha x)^{M-1}\end{pmatrix} + \begin{pmatrix}0\\ 0\\ \vdots\\ \varphi_{M+1}(x)\frac{1}{s_{M+1}\beta\alpha}\end{pmatrix}. \tag{5.14}$$

Note that the $1/s_M$ term that would appear in the vector to the right absorbed the existing $\sqrt{M/2}$ to become a $1/s_{M+1}$.

As described in [81], this relation is satisfied by a unique lower Hessenberg matrix, allowing us to conclude that

$$\mathsf{L}^{-1}\mathsf{G}^{-1}\mathsf{TGL} = F(H_M)\frac{1}{\beta\alpha}, \tag{5.15}$$

where $F(H_M)$ is the Frobenius matrix [136] associated with H_M. By definition,

$$F(H_M) = \mathsf{Z}^T - \boldsymbol{e}_M\boldsymbol{h}_M^T, \tag{5.16}$$

where

$$\boldsymbol{h}_M^T = \begin{pmatrix}\frac{h_0}{h_M} & \frac{h_1}{h_M} & \cdots & \frac{h_{M-2}}{h_M} & \frac{h_{M-1}}{h_M}\end{pmatrix}$$

is defined using the coefficients of the Hermite polynomial $H_M(x) = \sum_{k=0}^{M} h_k x^k$. The structure of this $F(H_M)$ matrix is

$$F(H_M) = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1\\ \frac{h_0}{h_M} & \frac{h_1}{h_M} & \frac{h_2}{h_M} & \cdots & \frac{h_{M-1}}{h_M}\end{pmatrix},$$

which helps explain why it is relevant. When applied to one column of a Vandermonde matrix, such as the vectors in (5.14), it yields

$$\begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ \frac{h_0}{h_M} & \frac{h_1}{h_M} & \frac{h_2}{h_M} & \dots & \frac{h_{M-1}}{h_M} \end{pmatrix} \begin{pmatrix} 1 \\ x \\ \vdots \\ x^{M-2} \\ x^{M-1} \end{pmatrix} = \begin{pmatrix} x \\ x^2 \\ \vdots \\ x^{M-1} \\ \frac{1}{h_M} H_{M-1}(x) \end{pmatrix}.$$

The compact design of $F(H_M)$ is a result of the three-term recurrence, where the "shift" effect seen here allows us to describe $x(\beta\alpha x)^k$ in terms of $(\beta\alpha x)^{k+1}$ except for $k = M-1$. Because $(\beta\alpha x)^M$ is not present in the vector, the final term needs to be handled directly with the three-term recurrence, rather than just the shift, necessitating the rank-one correction on the right side of (5.14).

Some matrix manipulations involving (5.15) and (5.16) produce a useful result:

$$\begin{aligned} \mathsf{L}^{-1}\mathsf{G}^{-1}\mathsf{T}\mathsf{G}\mathsf{L} &= (\mathsf{Z}^T - \boldsymbol{e}_M \boldsymbol{h}_M^T)\frac{1}{\beta\alpha}, \\ \beta\alpha\mathsf{T} &= \mathsf{G}\mathsf{L}\mathsf{Z}^T\mathsf{L}^{-1}\mathsf{G}^{-1} - \mathsf{G}\mathsf{L}\boldsymbol{e}_M\boldsymbol{h}_M^T\mathsf{L}^{-1}\mathsf{G}^{-1}, \\ \beta\alpha\mathsf{T}^T + \boldsymbol{\rho}\boldsymbol{e}_M^T &= \mathsf{G}^{-1}\mathsf{L}^{-T}\mathsf{Z}\mathsf{L}^T\mathsf{G}, \end{aligned} \tag{5.17}$$

where

$$\boldsymbol{\rho} = s_M \mathsf{G}^{-1}\mathsf{L}^{-T}\boldsymbol{h}_M.$$

Using (5.17) in (5.12) yields

$$\boldsymbol{u}_{k+1} = (\beta\alpha\mathsf{T}^T + \boldsymbol{\rho}\boldsymbol{e}_M^T + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\boldsymbol{u}_k - \frac{2b_k}{\sqrt{k^2-k}}\boldsymbol{u}_{k-1}.$$

We know that

$$\boldsymbol{\rho}\boldsymbol{e}_M^T\boldsymbol{u}_k = 0, \qquad 1 \le k \le M-1, \tag{5.18}$$

because U is upper triangular. This, combined with the symmetry of T, produces the final recurrence

$$\boldsymbol{u}_{k+1} = (\beta\alpha\mathsf{T} + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\boldsymbol{u}_k - \frac{2b_k}{\sqrt{k^2-k}}\boldsymbol{u}_{k-1}, \qquad 1 \le k \le M-1, \tag{5.19}$$

with $\boldsymbol{u}_{-1} = 0$.

5.2.2 Computing the diagonal factor

Now we have the ability to compute U^{-1}, and we must consider the rest of the (5.4) decomposition. The values of Σ can be recovered by exploiting the LDL factorization (5.9) to get

$$\mathsf{U}^{-T}\Phi^T\Phi\mathsf{U}^{-1} = \Sigma.$$

Considering this for each column of U exposes the orthogonality condition

$$\boldsymbol{u}_k^T\Phi^T\Phi\boldsymbol{u}_\ell = \boldsymbol{v}_k^T\boldsymbol{u}_\ell = \begin{cases} \sigma_k^2, & k = \ell \\ 0, & k \neq \ell \end{cases}, \tag{5.20}$$

where we define $\boldsymbol{v}_k = \Phi^T\Phi\boldsymbol{u}_k$. This provides a method for computing σ_k given $\boldsymbol{u}_k$.

Computing $\boldsymbol{v}_k$ at each step of the recurrence is unacceptably expensive, so we benefit by producing a recurrence for $\boldsymbol{v}_k$. Premultiplying (5.19) by $\Phi^T\Phi$ gives

$$\boldsymbol{v}_{k+1} = c_{k+1}\sqrt{\frac{2}{k}}\boldsymbol{v}_k - \frac{2b_k}{\sqrt{k^2-k}}\boldsymbol{v}_{k-1} + \beta\alpha\sqrt{\frac{2}{k}}\Phi^T\Phi\mathsf{T}\boldsymbol{u}_k, \quad 1 \le k \le M-1. \tag{5.21}$$

At this point, we use the identity

$$\Phi^T\Phi\mathsf{T} = \mathsf{T}\Phi^T\Phi + \boldsymbol{\delta}\boldsymbol{e}_M^T - \boldsymbol{e}_M\boldsymbol{\delta}^T, \tag{5.22}$$

derived in [81]. This identity is built upon the low displacement rank [102] of $\Phi^T\Phi$ with respect to the operator

$$\Delta_{\mathsf{T},\mathsf{T}}(\mathsf{X}) = \mathsf{T}\mathsf{X} - \mathsf{X}\mathsf{T}.$$

In this case, $\text{rank}(\Delta_{\mathsf{T},\mathsf{T}}(\Phi^T\Phi)) = 2$, and the specific nature of that low rank matrix was discussed in [81]. $(\boldsymbol{\delta})_k = \delta_k$ can be determined by considering $\Delta_{\mathsf{T},\mathsf{T}}(\Phi^T\Phi)\boldsymbol{e}_M$:

$$\begin{aligned}
(\mathsf{T}\Phi^T\Phi - \Phi^T\Phi\mathsf{T})\boldsymbol{e}_M &= (\boldsymbol{\delta}\boldsymbol{e}_M^T - \boldsymbol{e}_M\boldsymbol{\delta}^T)\boldsymbol{e}_M, \\
\mathsf{T}\Phi^T\Phi\boldsymbol{e}_M - \Phi^T\Phi\mathsf{T}\boldsymbol{e}_M &= \boldsymbol{\delta} - \boldsymbol{e}_M\boldsymbol{\delta}^T\boldsymbol{e}_M, \\
\mathsf{T}\Phi^T\Phi\boldsymbol{e}_M - \Phi^T\Phi\mathsf{T}\boldsymbol{e}_M &= \boldsymbol{\delta} - \delta_M\boldsymbol{e}_M.
\end{aligned}$$

At this point, we realize that the first $M-1$ values of $\boldsymbol{\delta}$ can be determined by

$$\boldsymbol{\delta} = \Phi^T\Phi\mathsf{T}\boldsymbol{e}_M - \mathsf{T}\Phi^T\Phi\boldsymbol{e}_M.$$

The Mth term in $\boldsymbol{\delta}$ can arbitrarily be set to 0 because it is never needed. This can be verified by considering the use of (5.22) in (5.21):

$$\begin{aligned}
\Phi^T\Phi\mathsf{T}\boldsymbol{u}_k &= (\mathsf{T}\Phi^T\Phi + \boldsymbol{\delta}\boldsymbol{e}_M^T - \boldsymbol{e}_M\boldsymbol{\delta}^T)\boldsymbol{u}_k, \\
\Phi^T\Phi\mathsf{T}\boldsymbol{u}_k &= \mathsf{T}\Phi^T\Phi\boldsymbol{u}_k + \boldsymbol{\delta}\underbrace{\boldsymbol{e}_M^T\boldsymbol{u}_k}_{=0} - \boldsymbol{e}_M\boldsymbol{\delta}^T\boldsymbol{u}_k, \\
\Phi^T\Phi\mathsf{T}\boldsymbol{u}_k &= \mathsf{T}\Phi^T\Phi\boldsymbol{u}_k - \boldsymbol{e}_M\boldsymbol{\delta}^T\boldsymbol{u}_k.
\end{aligned}$$

Because $\boldsymbol{u}_k$ has 0 for its Mth value the inner product $\boldsymbol{\delta}^T\boldsymbol{u}_k$ never involves δ_M, and thus it can be set to anything. Using this, and fixing $\boldsymbol{v}_{-1} = 0$, we can simplify the $\boldsymbol{v}$ recurrence to

$$\boldsymbol{v}_{k+1} = (\beta\alpha\mathsf{T} + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\boldsymbol{v}_k - \frac{2b_k}{\sqrt{k^2-k}}\boldsymbol{v}_{k-1} - \beta\alpha\sqrt{\frac{2}{k}}\boldsymbol{e}_M\boldsymbol{\delta}^T\boldsymbol{u}_k, \quad 1 \le k \le M-1.$$

The terms c_{k+1} and b_k can be determined by using the orthogonality condition (5.20). First, premultiplying (5.19) by $\boldsymbol{v}_k^T$ gives

$$\begin{aligned}
\boldsymbol{v}_k^T\boldsymbol{u}_{k+1} &= \boldsymbol{v}_k^T(\beta\alpha\mathsf{T} + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\boldsymbol{u}_k - \frac{2b_k}{\sqrt{k^2-k}}\boldsymbol{v}_k^T\boldsymbol{u}_{k-1}, \\
0 &= \boldsymbol{v}_k^T(\beta\alpha\mathsf{T} + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\boldsymbol{u}_k, \\
c_{k+1}\boldsymbol{v}_k^T\boldsymbol{u}_k &= -\beta\alpha\boldsymbol{v}_k^T\mathsf{T}\boldsymbol{u}_k, \\
c_{k+1} &= -\frac{\beta\alpha}{\sigma_k^2}\boldsymbol{v}_k^T\mathsf{T}\boldsymbol{u}_k. \qquad (5.23)
\end{aligned}$$

Now we can premultiply (5.19) by $\boldsymbol{v}_{k+1}^T$ to produce a useful identity

$$\begin{aligned}
\boldsymbol{v}_{k+1}^T\boldsymbol{u}_{k+1} &= \boldsymbol{v}_{k+1}^T(\beta\alpha\mathsf{T}+c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\boldsymbol{u}_k - \frac{2b_k}{\sqrt{k^2-k}}\boldsymbol{v}_{k+1}^T\boldsymbol{u}_{k-1},\\
\sigma_{k+1}^2 &= \beta\alpha\sqrt{\frac{2}{k}}\boldsymbol{v}_{k+1}^T\mathsf{T}\boldsymbol{u}_k,\\
\frac{\sigma_{k+1}^2}{\beta\alpha}\sqrt{\frac{k}{2}} &= \boldsymbol{v}_{k+1}^T\mathsf{T}\boldsymbol{u}_k.
\end{aligned} \tag{5.24}$$

To determine b_k we premultiply (5.19) by $\boldsymbol{v}_{k-1}^T$:

$$\begin{aligned}
\boldsymbol{v}_{k-1}^T\boldsymbol{u}_{k+1} &= \boldsymbol{v}_{k-1}^T(\beta\alpha\mathsf{T}+c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\boldsymbol{u}_k - \frac{2b_k}{\sqrt{k^2-k}}\boldsymbol{v}_{k-1}^T\boldsymbol{u}_{k-1},\\
0 &= \beta\alpha\sqrt{\frac{2}{k}}\boldsymbol{v}_{k-1}^T\mathsf{T}\boldsymbol{u}_k - \frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1}^2,\\
b_k &= \frac{\beta\alpha}{\sigma_{k-1}^2}\sqrt{\frac{k-1}{2}}\boldsymbol{v}_{k-1}^T\mathsf{T}\boldsymbol{u}_k.
\end{aligned} \tag{5.25}$$

It is possible to further simplify $\boldsymbol{v}_{k-1}^T\mathsf{T}\boldsymbol{u}_k$, first by exploiting (5.22) to produce

$$\begin{aligned}
\boldsymbol{v}_{k-1}^T\mathsf{T}\boldsymbol{u}_k &= \boldsymbol{u}_{k-1}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\mathsf{T}\boldsymbol{u}_k,\\
&= \boldsymbol{u}_{k-1}^T(\mathsf{T}\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\boldsymbol{\delta}\boldsymbol{e}_M^T-\boldsymbol{e}_M\boldsymbol{\delta}^T)\boldsymbol{u}_k,\\
&= \boldsymbol{u}_{k-1}^T\mathsf{T}\underbrace{\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{u}_k}_{\boldsymbol{v}_k}+\boldsymbol{u}_{k-1}^T\boldsymbol{\delta}\underbrace{\boldsymbol{e}_M^T\boldsymbol{u}_k}_{=0}-\underbrace{\boldsymbol{u}_{k-1}^T\boldsymbol{e}_M}_{=0}\boldsymbol{\delta}^T\boldsymbol{u}_k,\\
&= \boldsymbol{u}_{k-1}^T\mathsf{T}\boldsymbol{v}_k = \boldsymbol{v}_k\mathsf{T}\boldsymbol{u}_{k-1}^T,
\end{aligned}$$

where the final line is valid because T is symmetric. Now we can substitute (5.24) to get

$$\begin{aligned}
\boldsymbol{v}_{k-1}^T\mathsf{T}\boldsymbol{u}_k &= \boldsymbol{v}_k\mathsf{T}\boldsymbol{u}_{k-1}^T\\
&= \frac{\sigma_k^2}{\beta\alpha}\sqrt{\frac{k-1}{2}}
\end{aligned}$$

which, when used in (5.26), allows us to write

$$b_k = \frac{k-1}{2}\left(\frac{\sigma_k}{\sigma_{k-1}}\right)^2. \tag{5.26}$$

5.2.3 Computing the orthogonal factor

The final necessary piece of the least squares solver is to compute the Q component of (5.4). That equation can be rewritten as $\mathsf{\Phi U}^{-1} = \mathsf{Q\Sigma}$, or at the column level,

$$\sigma_k \boldsymbol{q}_k = \mathsf{\Phi} \boldsymbol{u}_k.$$

To leverage (5.19) in computing Q, we premultiply by $\mathsf{\Phi}$ and make that column level substitution

$$\sigma_{k+1}\boldsymbol{q}_{k+1} = \beta\alpha\sqrt{\frac{2}{k}}\mathsf{\Phi T}\boldsymbol{u}_k + c_{k+1}\sqrt{\frac{2}{k}}\sigma_k \boldsymbol{q}_k - \frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1}\boldsymbol{q}_{k-1}. \tag{5.27}$$

Now $\mathsf{\Phi T}\boldsymbol{u}_k$ must be considered. We start with the result from [81],

$$\mathsf{H}\tilde{\mathsf{T}}^T = \beta\alpha\mathsf{D}_x\mathsf{H} + \boldsymbol{\omega}\boldsymbol{e}_M^T, \tag{5.28}$$

where $\boldsymbol{\omega}$ is an as yet unknown vector related to the values of $\boldsymbol{x}$, and

$$\tilde{\mathsf{T}} = \begin{pmatrix} 0 & .5 & & & \\ 1 & 0 & .5 & & \\ & 2 & 0 & \ddots & \\ & & \ddots & \ddots & .5 \\ & & & M-1 & 0 \end{pmatrix}, \qquad \mathsf{D}_x = \begin{pmatrix} x_1 & & & & \\ & x_2 & & & \\ & & \ddots & & \\ & & & x_{N-1} & \\ & & & & x_N \end{pmatrix}.$$

The matrix $\tilde{\mathsf{T}}$ is the analog of T for the Hermite polynomials, which is to say that it is derived from the associated Hermite three-term recurrence (5.5). Note that we can determine the displacement rank of H by writing

$$\begin{aligned} \Delta_{\beta\alpha\mathsf{D}_x,\tilde{\mathsf{T}}^T}(\mathsf{H}) &= \beta\alpha\mathsf{D}_x\mathsf{H} - \mathsf{H}\tilde{\mathsf{T}}^T \\ &= -\boldsymbol{\omega}\boldsymbol{e}_M^T. \end{aligned}$$

Therefore, we know that $\operatorname{rank}(\Delta_{\beta\alpha\mathsf{D}_x,\tilde{\mathsf{T}}^T}(\mathsf{H})) = 1$, which allows for the fast decomposition for Hermite polynomials. We want to determine if there are matrices A

and B so that

$$\operatorname{rank}(\Delta_{\mathsf{A},\mathsf{B}}(\Phi)) = 1.$$

Finding A and B is necessary to convert the $\Phi\mathsf{T}\boldsymbol{u}_k$ term in (5.27) to something which can be computed cheaply.

We notice that there is a distinct relationship between the T and $\tilde{\mathsf{T}}$ matrices, and we can exploit that to determine the A and B matrices. The relationship happens to be

$$\mathsf{G}\tilde{\mathsf{T}} = \mathsf{T}\mathsf{G}\beta\alpha, \tag{5.29}$$

which can be confirmed by studying the subdiagonals and superdiagonals (since everything else is zero),

$\mathsf{G}\tilde{\mathsf{T}} = \mathsf{T}\mathsf{G}\beta\alpha$, row k, superdiagonal:

$$\gamma_k(.5) = \gamma_{k+1}\left(\frac{\sqrt{k+1}}{\sqrt{2}\beta\alpha}\right)\beta\alpha,$$

$$\frac{\gamma_k}{\gamma_{k+1}} = \sqrt{2(k+1)}, \qquad \checkmark$$

$\mathsf{G}\tilde{\mathsf{T}} = \mathsf{T}\mathsf{G}\beta\alpha$, row k, subdiagonal:

$$\gamma_k(k-1) = \gamma_{k-1}\left(\frac{\sqrt{k-1}}{\sqrt{2}\beta\alpha}\right)\beta\alpha,$$

$$\frac{\gamma_k}{\gamma_{k-1}} = \frac{1}{\sqrt{2(k-1)}}. \qquad \checkmark$$

Starting with (5.28) we premultiply by D and postmultiply by G to find

$$\mathsf{H}\tilde{\mathsf{T}}^T = \beta\alpha\mathsf{D}_x\mathsf{H} + \boldsymbol{\omega}\boldsymbol{e}_M^T,$$

$$\mathsf{D}\mathsf{H}\tilde{\mathsf{T}}^T\mathsf{G} = \mathsf{D}\beta\alpha\mathsf{D}_x\mathsf{H}\mathsf{G} + \mathsf{D}\boldsymbol{\omega}\boldsymbol{e}_M^T\mathsf{G}.$$

Now we use the transpose of (5.29) to convert the $\tilde{\mathsf{T}}$ matrix to T,

$$\mathsf{D}\mathsf{H}\mathsf{G}\mathsf{T}\beta\alpha = \mathsf{D}\beta\alpha\mathsf{D}_x\mathsf{H}\mathsf{G} + \mathsf{D}\boldsymbol{\omega}\boldsymbol{e}_M^T\mathsf{G}.$$

Swapping diagonal matrices D and D_x, and substiting (5.7) gives us the needed result

$$\mathsf{\Phi T} = \mathsf{D}_x\mathsf{\Phi} + \bar{\boldsymbol{\omega}}\boldsymbol{e}_M^T,$$

where $\bar{\boldsymbol{\omega}} = \gamma_M \mathsf{D}\boldsymbol{\omega}/(\beta\alpha)$. Using this in (5.27) produces

$$\begin{aligned}\sigma_{k+1}\boldsymbol{q}_{k+1} &= \beta\alpha\sqrt{\frac{2}{k}}(\mathsf{D}_x\mathsf{\Phi} + \bar{\boldsymbol{\omega}}\boldsymbol{e}_M^T)\boldsymbol{u}_k + c_{k+1}\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k - \frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1}\boldsymbol{q}_{k-1}. \\ &= \beta\alpha\sqrt{\frac{2}{k}}\mathsf{D}_x\mathsf{\Phi}\boldsymbol{u}_k + \beta\alpha\sqrt{\frac{2}{k}}\bar{\boldsymbol{\omega}}\underbrace{\boldsymbol{e}_M^T\boldsymbol{u}_k}_{=0} + c_{k+1}\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k - \frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1}\boldsymbol{q}_{k-1}.\end{aligned}$$

Replacing $\mathsf{\Phi}\boldsymbol{u}_k$ with $\boldsymbol{q}_k$ produces the desired recurrence,

$$\sigma_{k+1}\boldsymbol{q}_{k+1} = (\beta\alpha\mathsf{D}_x + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k - \frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1}\boldsymbol{q}_{k-1}, \quad 1 \le k \le M-1. \tag{5.30}$$

The first column $\boldsymbol{q}_0$ is chosen to be the normalized first eigenfunction,

$$\boldsymbol{q}_0 = \frac{\boldsymbol{\phi}_1(\boldsymbol{x})}{\|\boldsymbol{\phi}_1(\boldsymbol{x})\|_2},$$

which is the first column of $\mathsf{\Phi}$, the standard choice for a QR factorization.

5.3 A fast QR algorithm for eigenfunctions

Below is the algorithm to compute $\mathsf{\Phi} = \mathsf{Q\Sigma U}$, where $\mathsf{Q} \in \mathbb{R}^{N\times M}$ has orthonormal columns, $\mathsf{\Sigma} \in \mathbb{R}^{M\times M}$ is diagonal with positive entries, and $\mathsf{U} \in \mathbb{R}^{M\times M}$ is upper triangular with unit diagonal. This algorithm is called the U-centric fast QR decomposition because the iterative updates b, c and σ_k are computed using $\boldsymbol{u}_k$, which must be computed first. In Section 5.4 we reconsider this choice by deriving a factorization which is Q-centric, which lends relevance to this distinction. Note that the algorithm is split in to *initialization* and *iteration* components.

Algorithm 3.a U-centric fast QR decomposition: *intialization*

Given $\boldsymbol{x}$, Φ, β, α
$\boldsymbol{p}_1 \leftarrow \Phi^T \boldsymbol{\phi}_1(\boldsymbol{x})$, $\boldsymbol{p}_{M-1} \leftarrow \Phi^T \boldsymbol{\phi}_{M-1}(\boldsymbol{x})$, $\boldsymbol{p}_M \leftarrow \Phi^T \boldsymbol{\phi}_M(\boldsymbol{x})$
$\boldsymbol{\delta} \leftarrow \sqrt{M-1}/(\sqrt{2}\beta\alpha)\boldsymbol{p}_{M-1} - \mathsf{T}\boldsymbol{p}_M$, $\mathsf{D}_x \leftarrow \mathrm{diag}(\boldsymbol{x})$
$\boldsymbol{u}_1 \leftarrow \boldsymbol{e}_1$, $\boldsymbol{v}_1 \leftarrow \boldsymbol{p}_1$, $\sigma_1 \leftarrow \sqrt{\boldsymbol{v}_1^T \boldsymbol{u}_1}$, $\boldsymbol{q}_1 \leftarrow \varphi_1(\boldsymbol{x})^T/\sigma_1$
$c \leftarrow -(\beta\alpha/\sigma_1^2)\boldsymbol{v}_1^T \mathsf{T}\boldsymbol{u}_1$
$\boldsymbol{u}_2 \leftarrow (\beta\alpha \mathsf{T}\boldsymbol{u}_1 + c\boldsymbol{u}_1)\sqrt{2}$
$\boldsymbol{v}_2 \leftarrow (\beta\alpha(\mathsf{T}\boldsymbol{v}_1 - \boldsymbol{\delta}^T \boldsymbol{u}_1 \boldsymbol{e}_M) + c\boldsymbol{v}_1)\sqrt{2}$
$\sigma_2 \leftarrow \sqrt{\boldsymbol{v}_2^T \boldsymbol{u}_2}$
$\boldsymbol{q}_2 \leftarrow (\beta\alpha \mathsf{D}_x \boldsymbol{q}_1 + c\boldsymbol{q}_1)\sqrt{2}(\sigma_1/\sigma_2)$
return $\boldsymbol{u}_1$, $\boldsymbol{v}_1$, σ_1, $\boldsymbol{q}_1$, $\boldsymbol{u}_2$, $\boldsymbol{v}_2$, σ_2, $\boldsymbol{q}_2$, $\boldsymbol{\delta}$, D_x

Algorithm 3.b U-centric fast QR decomposition: *iteration*

Execute **Algorithm 3.a**
for $k = 2$ to $M-1$ **do**
  $b \leftarrow (\sigma_k/\sigma_{k-1})^2(k-1)/2$
  $c \leftarrow -(\beta\alpha/\sigma_k^2)\boldsymbol{v}_k^T \mathsf{T}\boldsymbol{u}_k$
  $\boldsymbol{u}_{k+1} \leftarrow \sqrt{2/k}(\beta\alpha \mathsf{T}\boldsymbol{u}_k + c\boldsymbol{u}_k) - 2b/\sqrt{k^2-k}\boldsymbol{u}_{k-1}$
  $\boldsymbol{v}_{k+1} \leftarrow (\beta\alpha(\mathsf{T}\boldsymbol{v}_k - \boldsymbol{\delta}^T \boldsymbol{u}_k \boldsymbol{e}_M) + c\boldsymbol{v}_k)\sqrt{2/k} - 2b/\sqrt{k^2-k}\boldsymbol{v}_{k-1}$
  $\sigma_{k+1} \leftarrow \sqrt{\boldsymbol{v}_{k+1}^T \boldsymbol{u}_{k+1}}$
  $\boldsymbol{q}_{k+1} \leftarrow (\beta\alpha \mathsf{D}_x \boldsymbol{q}_k + c\boldsymbol{q}_k)\sqrt{2/k}(\sigma_k/\sigma_{k+1}) - (2b/\sqrt{k^2-k})(\sigma_{k-1}/\sigma_{k+1})\boldsymbol{q}_{k-1}$
end for
return U, Σ, Q

It should be noted that U^{-1} can be computed without computing Q; all the $\boldsymbol{u}_k$ computations are independent of the values of $\boldsymbol{q}_k$. This is useful because (5.3) can be converted to its associated normal equations by premultiplying by Φ^T,

$$\Phi \boldsymbol{c} = \boldsymbol{y},$$
$$\Phi^T \Phi \boldsymbol{c} = \Phi^T \boldsymbol{y},$$
$$\mathsf{U}^T \Sigma^2 \mathsf{U} \boldsymbol{c} = \tilde{\boldsymbol{y}}.$$

The third line invokes (5.9) to produce the LDL factorization, and redefines the right hand side to $\tilde{\boldsymbol{y}} = \Phi^T \boldsymbol{y}$. This system is now a square $M \times M$ system, which may be a valid solution technique depending on the condition of the Φ system.

To ensure that the algorithm indeed performs better than generic QR, consider the following cost analysis. Bear in mind that this should only be considered as a statement of the order of magnitude, as the actual number of flops will vary based on the implementation. In flops, the initialization phase incurs

- $6MN$ for finding $\boldsymbol{p}$, the first and last two columns of $\Phi^T \Phi$,
- $5M$ for finding $\boldsymbol{\delta}$,
- $2M + N$ for finding $\boldsymbol{u}_1$, $\boldsymbol{v}_1$, σ_1 and $\boldsymbol{q}_1$, and
- $20M + 4N$ for finding c, $\boldsymbol{u}_2$, $\boldsymbol{v}_2$, σ_2, $\boldsymbol{q}_2$,

which amounts to $6MN + 5N + 27M$ total initialization flops. Each iteration of the algorithm requires

- $6M$ to find b and c, and
- $20M + 6N$to find $\boldsymbol{u}_{k+1}$, $\boldsymbol{v}_{k+1}$, σ_{k+1} and $\boldsymbol{q}_{k+1}$.

Since this algorithm iterates $M-2$ times, the cost of the iterations is $6NM + 20M^2 - 12N - 34M$. Asymptotically, the cost of this fast QR method is roughly $12NM + 20M^2$, which is indeed faster than the $\mathcal{O}(NM^2)$ generally associated with QR factorizations.

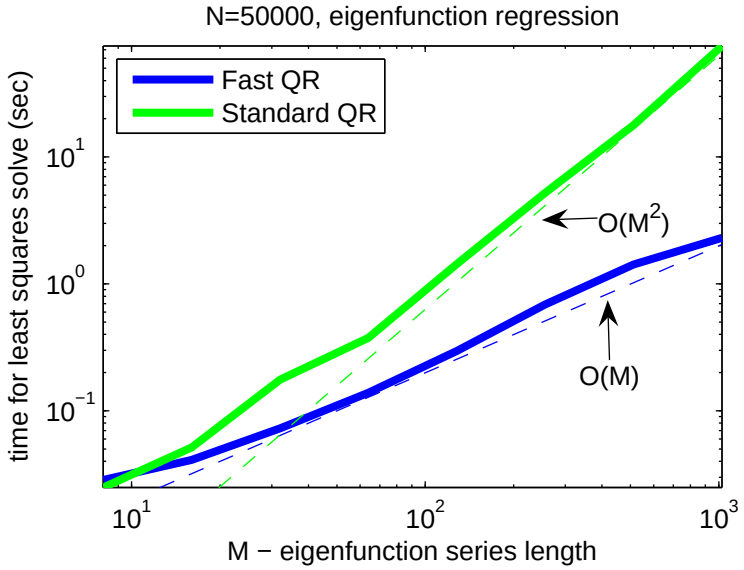


Figure 5.1: For N fixed at 50000, the fast QR algorithm performs better than standard QR when $M > 8$.

Figure 5.1 shows the timing results from a numerical experiment solving (5.3) when $N = 50000$ and M varies between 8 and 1024. The fast QR method has $\mathcal{O}(M)$ complexity, while the standard QR method available in MATLAB appears near $\mathcal{O}(M^2)$ complexity. When $N \gg M$ (i.e., the left of the figure,) the dominant cost for fast QR and MATLAB QR is roughly of the same order, but the coefficient for fast QR is 12, whereas it is only 4/3 for standard QR. This explains why the fast and standard methods are comparable when M is very small.

5.4 Reconsidering the fast QR decomposition

The driving force in the QR factorization from Section 5.3 is the decomposition of U^{-1}: c_k and σ_k (and therefore also b_{k-1}) are computed from $\boldsymbol{u}_k$, whereas $\boldsymbol{q}_k$ is updated using those values. This need not be the case, and in this section we derive an algorithm using $\boldsymbol{q}_k$ as the foundation of the decomposition. Starting from (5.30), we can determine b and c using $\mathsf{Q}^T\mathsf{Q} = \mathsf{I}_M$. First, premultiplying by $\boldsymbol{q}_k^T$ gives

$$\begin{aligned}
\sigma_{k+1}\boldsymbol{q}_k^T\boldsymbol{q}_{k+1} &= \boldsymbol{q}_k^T(\beta\alpha\mathsf{D}_x + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k - \frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1}\boldsymbol{q}_k^T\boldsymbol{q}_{k-1},\\
0 &= \boldsymbol{q}_k^T(\beta\alpha\mathsf{D}_x + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k,\\
\boldsymbol{q}_k^T c_{k+1}\mathsf{I}_M\boldsymbol{q}_k &= -\boldsymbol{q}_k^T\beta\alpha\mathsf{D}_x\boldsymbol{q}_k,\\
c_{k+1} &= -\beta\alpha\boldsymbol{q}_k^T\mathsf{D}_x\boldsymbol{q}_k. \qquad (5.31)
\end{aligned}$$

Premultiplying by $\boldsymbol{q}_{k-1}^T$ gives

$$\begin{aligned}
\sigma_{k+1}\boldsymbol{q}_{k-1}^T\boldsymbol{q}_{k+1} &= \boldsymbol{q}_{k-1}^T(\beta\alpha\mathsf{D}_x + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k - \frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1}\boldsymbol{q}_{k-1}^T\boldsymbol{q}_{k-1},\\
\frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1} &= \boldsymbol{q}_{k-1}^T(\beta\alpha\mathsf{D}_x + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k,\\
\sqrt{2}b_k\sigma_{k-1} &= \boldsymbol{q}_{k-1}^T\beta\alpha\mathsf{D}_x\sqrt{k-1}\sigma_k\boldsymbol{q}_k,\\
b_k &= \beta\alpha\sqrt{\frac{k-1}{2}}\frac{\sigma_k}{\sigma_{k-1}}\boldsymbol{q}_{k-1}^T\mathsf{D}_x\boldsymbol{q}_k.
\end{aligned}$$

We can simplify this by premultiplying (5.30) by $\boldsymbol{q}_{k+1}^T$

$$\begin{aligned}
\sigma_{k+1}\boldsymbol{q}_{k+1}^T\boldsymbol{q}_{k+1} &= \boldsymbol{q}_{k+1}^T(\beta\alpha\mathsf{D}_x + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k - \frac{2b_k}{\sqrt{k^2-k}}\sigma_{k-1}\boldsymbol{q}_{k+1}^T\boldsymbol{q}_{k-1},\\
\sigma_{k+1} &= \boldsymbol{q}_{k+1}^T(\beta\alpha\mathsf{D}_x + c_{k+1}\mathsf{I}_M)\sqrt{\frac{2}{k}}\sigma_k\boldsymbol{q}_k,\\
\frac{1}{\beta\alpha}\frac{\sigma_{k+1}}{\sigma_k}\sqrt{\frac{k}{2}} &= \boldsymbol{q}_{k+1}^T\mathsf{D}_x\boldsymbol{q}_k.
\end{aligned}$$

Using this identity we can finally define b_k as

$$\begin{aligned} b_k &= \beta\alpha\sqrt{\frac{k-1}{2}}\frac{\sigma_k}{\sigma_{k-1}}\left(\frac{1}{\beta\alpha}\frac{\sigma_k}{\sigma_{k-1}}\sqrt{\frac{k-1}{2}}\right) \\ &= \frac{k-1}{2}\left(\frac{\sigma_k}{\sigma_{k-1}}\right)^2, \end{aligned}$$

which is (fortunately) the same value as was derived in (5.26). Notice that by defining the problem through $\boldsymbol{q}_k$ rather than $\boldsymbol{u}_k$, we no longer need to keep track of $\boldsymbol{v}_k$.

5.4.1 A fast QR algorithm through $\boldsymbol{q}_k$

In Section 5.3 the fast QR algorithm computed σ_k using the columns $\boldsymbol{u}_k$. Here we present the algorithm which instead computes σ_k using the columns $\boldsymbol{q}_k$. The structure of this algorithm is very similar, but it has significant advantages which are discussed.

Algorithm 4.a Q-centric fast QR decomposition: *intialization*

Given $\boldsymbol{x}$, β, α
$\mathsf{D}_x \leftarrow \text{diag}(\boldsymbol{x})$
$\boldsymbol{u}_1 \leftarrow \boldsymbol{e}_1$, $\sigma_1 \leftarrow \|\boldsymbol{\phi}_1(\boldsymbol{x})\|_2$, $\boldsymbol{q}_1 \leftarrow \boldsymbol{\phi}_1(\boldsymbol{x})^T/\sigma_1$
$c \leftarrow -\beta\alpha\boldsymbol{q}_1^T\mathsf{D}_x\boldsymbol{q}_1$
$\boldsymbol{p} \leftarrow \sqrt{2}(\beta\alpha\mathsf{D}_x + c\mathsf{I}_N)\sigma_1\boldsymbol{q}_1$
$\sigma_2 \leftarrow \|\boldsymbol{p}\|_2$
$\boldsymbol{q}_2 \leftarrow \boldsymbol{p}/\sigma_2$
$\boldsymbol{u}_2 \leftarrow \sqrt{2}(\beta\alpha\mathsf{T} + c\mathsf{I}_M)\boldsymbol{u}_1$
return $\boldsymbol{u}_1$, $\boldsymbol{q}_1$, $\boldsymbol{u}_2$, σ_2, $\boldsymbol{q}_2$, D_x

One of the main advantages of computing the decomposition using this Q centric approach is that $\boldsymbol{v}_k$ no longer needs to be calculated. This saves costs and memory throughout the iteration, and also avoids the initialization work associated with computing $\boldsymbol{\delta}$. The earlier algorithm did allow for a solution of the normal

Algorithm 4.b Q-centric fast QR decomposition: *iteration*

Execute **Algorithm 4.a**
for $k = 2$ to $M - 1$ **do**
  $b \leftarrow (\sigma_k/\sigma_{k-1})^2(k-1)/2$
  $c \leftarrow -\beta\alpha \boldsymbol{q}_k^T \mathsf{D}_x \boldsymbol{q}_k$
  $\boldsymbol{p} \leftarrow \sqrt{2/k}(\beta\alpha\mathsf{D}_x + c\mathsf{I}_N)\sigma_k\boldsymbol{q}_k - (2b/\sqrt{k^2-k})\sigma_{k-1}\boldsymbol{q}_{k-1}$
  $\sigma_{k+1} \leftarrow \|\boldsymbol{p}\|_2$
  $\boldsymbol{q}_{k+1} \leftarrow \boldsymbol{p}/\sigma_{k+1}$
  $\boldsymbol{u}_{k+1} \leftarrow \sqrt{2/k}(\beta\alpha\mathsf{T} + c\mathsf{I}_M)\boldsymbol{u}_k - (2b/\sqrt{k^2-k})\boldsymbol{u}_{k-1}$
end for
return U, Σ, Q

equations without computing the Q term, which is obviously not possible here if Q is used to propagate the iteration, so there is some tradeoff.

Another benefit to this approach is that M need not be fixed prior to beginning the factorization. In the earlier algorithm, $\boldsymbol{\phi}_M(\boldsymbol{x})$ was required to compute $\boldsymbol{\delta}$; for this Q centric algorithm there is no such requirement. This allows for the possibility of an adaptive algorithm, where the σ_k values are monitored and the factorization is truncated at $\sigma_M < \sigma_{\text{tol}}$. Doing this would perforate the insulation between the approximation scheme and the linear solver (by allowing the linear solver to dictate the appropriate complexity of the kernel-based approximation), but this may allow for less work overall.

5.4.2 Q-centric QR algorithm complexity analysis

The same complexity analysis conducted for the earlier fast QR algorithm can be used here to study the leading order cost. This algorithm assumes a cost of 5 flops to compute the first eigenfunction, which is not present in the earlier algorithm. There it was assumed that the user had already computed Φ, whereas here the algorithm computes the first eigenfunction. The 5 flop assumption is just

an estimate, and does not affect the final asymptotic complexity order.

In flops, the initialization phase incurs

- $8N$ for finding $\boldsymbol{u}_1$, $\boldsymbol{v}_1$, σ_1 and $\boldsymbol{q}_1$, and
- $9N + 6M$ for finding c, $\boldsymbol{u}_2$, $\boldsymbol{v}_2$, σ_2, $\boldsymbol{q}_2$,

which amounts to $17N + 6M$ total initialization flops. Each iteration of the algorithm requires

- $3N$ to find b and c, and
- $8N + 8M$ to find $\boldsymbol{u}_{k+1}$, $\boldsymbol{v}_{k+1}$, σ_{k+1} and $\boldsymbol{q}_{k+1}$.

Since this algorithm iterates $M - 2$ times, the cost of the iterations is $11NM + 8M^2 - 22N - 16M$. Asymptotically, the cost of this version of the fast QR method is roughly $11NM + 8M^2$, which is both faster than the $\mathcal{O}(NM^2)$ generally associated with QR factorizations, and slightly faster than the earlier algorithm.

5.4.3 Analysis of error for the fast QR algorithms

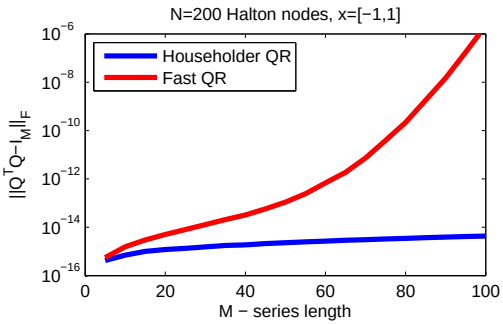
One of the common fears when computing orthonormal factorizations using non-orthogonal transformations (such as this three-term recurrence) is that round-off error will accumulate and orthonormality will be lost. One method we use to help determine the stability of these fast algorithms is to analyze the quantity

$$\Theta(M) = \|\mathsf{Q}^T\mathsf{Q} - \mathsf{I}_M\|_F, \tag{5.32}$$

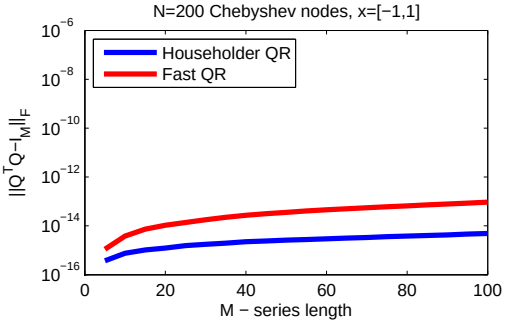
where Q is the orthonormal matrix from (5.4). If this value is 0, then Q is orthonormal, which should always be true for a Q produced via Householder reflections or some other orthogonal technique.

At this point, we note that the $\boldsymbol{q}_k$ centric algorithm is expected to produce better results for (5.32) for two reasons. The first is that the b and c values produced during the iteration are computed under the assumption that Q is orthonormal, which may help maintain orthogonality throughout the iteration. The second reason is that all $\|\boldsymbol{q}_k\|_2 = 1$ by design; $\boldsymbol{q}_k$ is normalized in step k, so only off diagonal terms contribute to (5.32). These benefits, as well as the reduced cost, lead us to limit our error study to only the $\boldsymbol{q}$ centric fast QR algorithm.

To study the value of $\Theta(M)$ we consider a numerical experiment. We place $N = 200$ points at the Halton nodes [180], and $N = 200$ points at the roots of the Chebyshev polynomial on the domain $x \in [-1, 1]$. The resulting values of $\Theta(M)$ are displayed in Figure 5.2a and Figure 5.2b respectively.



(a) On quasi-uniform points, orthogonality is lost quickly for $M > 50$.



(b) While not perfect, orthogonality is relatively well-preserved on the Chebyshev nodes.

Figure 5.2: The distribution of points is significant for preserving orthogonality. These graphs were produced with $\alpha = 1$ and $\varepsilon = .001$.

What is immediately obvious when viewing Figure 5.2 is that the fast QR algorithm loses some degree of orthogonality in all circumstances. The traditional QR decomposition (which requires $\mathcal{O}(M^2N)$ work) should be considered as a baseline,

as the best possible orthogonality which could be achieved. For the Chebyshev nodes, orthogonality is maintained rather well, suggesting that this fast QR approach is useful when we have the ability to choose the points at which the function is sampled. When studying the pseudo-randomly spaced Halton points, we see that orthogonality is lost at an exponential rate for $M > .25N$. This suggests that for scattered data locations, fast QR is only appropriate when M is small relative to N.

Loss of orthogonality is only one way to measure the viability of this fast QR algorithm. Perhaps the more appropriate way to study the viability of this method is to consider its accuracy when applied to an approximation problem. Consider again $N = 200$ points, this time sampling the two Bessel functions

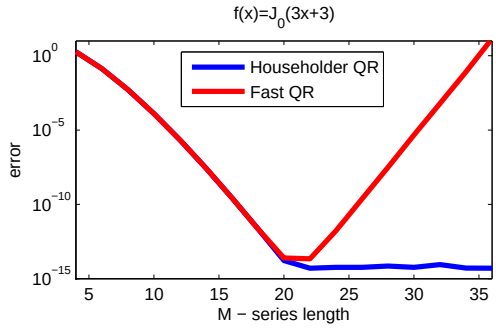
$$f_1(x) = J_0(3(x+1))$$
$$f_2(x) = J_0(6(x+1))$$

at the Halton nodes on the domain $x \in [-1, 1]$. In Figure 5.3 both traditional QR and fast QR solves are compared, with error computed at 1000 evenly spaced nodes throughout the domain.

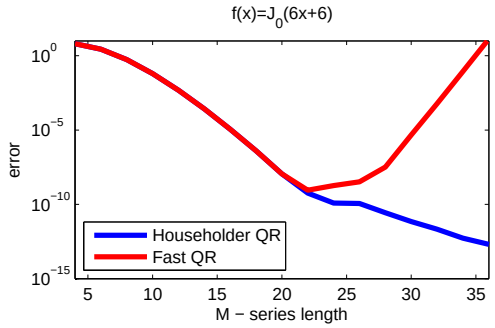
As we can see here, the quality of the fast QR solution is strongly dependent on the choice of M. The idea that there is an optimal M value for the approximation was discussed in Section 4.6.2, but the divergent behavior of the traditional and fast QR solutions indicates that we are instead seeing the effects of error. Specifically, the errors in Q become the dominant source of error for the solution.

To see this clearly, consider the iteration which produces the approximate solution $\boldsymbol{c}_k$ to $\Phi\boldsymbol{c} = \boldsymbol{y}$ at step k

$$\boldsymbol{c}_k = \boldsymbol{c}_{k-1} + \frac{\boldsymbol{q}_k^T \boldsymbol{y}}{\sigma_k} \boldsymbol{u}_k, \qquad k = 1, ..., M, \tag{5.33}$$



(a) The fast QR method can achieve roughly the same optimal error as the slow QR method.



(b) Because this function is more complex, the fast QR method is unable to achieve optimal accuracy.

Figure 5.3: The fast QR method mirrors the Householder QR, until the error in the Q computation degrades the solution. For sufficiently simple functions, this occurs for small M, and thus fast QR is appropriate. When the function requires higher M, the fast QR algorithm may produce too much error to be useful. These experiments used $\varepsilon = .001$ and $\alpha = 1$.

with an initial iterate $\boldsymbol{c}_0$ of all zeros. Note that the traditional QR factorization can be made equivalent to the fast QR factorization with small manipulations:

$$\begin{aligned}\Phi &= \mathsf{Q}_S\mathsf{R}_S \\ &= \mathsf{Q}_S\mathsf{D}_S\underbrace{\mathsf{D}_S^{-1}\mathsf{R}_S}_{=\mathsf{U}}\end{aligned}$$

where $\mathsf{D}_S^{-1} = \operatorname{diag}(\mathsf{R}_S)$. This is almost the $\Phi = \mathsf{Q}\Sigma\mathsf{U}$ decomposition, but we don't necessarily have positive values on the diagonal matrix. By defining $(\mathsf{P}_S)_{kk} = \operatorname{sign}((\mathsf{D}_S)_{kk})$ we can fix this ± 1 issue and write

$$\begin{aligned}\Phi &= \mathsf{Q}_S\mathsf{D}_S\mathsf{D}_S^{-1}\mathsf{R}_S \\ &= \underbrace{\mathsf{Q}_S\mathsf{P}_S}_{=\mathsf{Q}}\underbrace{\mathsf{P}_S\mathsf{D}_S}_{=\Sigma}\underbrace{\mathsf{D}_S^{-1}\mathsf{R}_S}_{=\mathsf{U}}\end{aligned}$$

Now that we can compare the provably stable Householder QR to the fast QR, examine Table 5.1, where the progress of the solution using (5.33) is monitored.

Table 5.1a shows that there is significant error in the calculation even before orthogonality was noticeably lost in Figure 5.2a. This by itself is not necessarily

k	Fast		Slow	
	$\boldsymbol{q}_k^T\boldsymbol{y}$	σ_k	$\boldsymbol{q}_k^T\boldsymbol{y}$	σ_k
19	1.285e-07	4.943e-10	1.285e-07	4.943e-10
20	9.608e-09	7.975e-11	9.610e-09	7.971e-11
21	3.037e-09	1.264e-11	3.029e-09	1.277e-11
22	1.968e-10	1.932e-12	1.882e-10	2.790e-12
23	5.771e-11	2.876e-13	2.658e-10	3.461e-12

(a) Incorrect digits have been underlined and put in red. The errors in the $\boldsymbol{q}_k$ computation have permeated the iteration and caused the factorization to be dominated by error.

k	$\boldsymbol{q}_k^T\boldsymbol{y}/\sigma_k$	
	Fast	Slow
7	1.356e+001	1.356e+001
14	1.047e+002	1.047e+002
21	2.402e+002	2.372e+002
28	5.971e+001	2.982e+000
35	1.330e+008	1.817e-001

(b) The magnitude of later terms is inappropriately large for fast QR, leading to instability.

Table 5.1: $N = 200$ Halton points are used to approximate $f_2(x)$. The fast QR algorithm is acceptable when M is sufficiently small. When M is too large, error dominates $\boldsymbol{q}_k^T\boldsymbol{y}$ which prevents the stability seen in slow QR.

a problem, because as long as orthogonality is reasonably well maintained, the factorization (5.4) is still valid. The problem caused by this error accumulation is more obvious in Table 5.1b, where rather than the reduced emphasis of higher order terms as seen for the slow QR, the higher order terms are fallaciously magnified with fast QR. This discrepancy exists because the fast QR algorithm causes $\sigma_k \to 0$, but $\boldsymbol{q}_k^T\boldsymbol{y} \to \epsilon_{\text{mach}}$ because of the error always present in $\boldsymbol{q}_k$. As a result, $\boldsymbol{q}_k^T\boldsymbol{y}/\sigma_k \to \infty$ after the error accumulation dominates, which happens in Table 5.1a for $k \approx 20$.

5.4.4 An adaptive, fast least-squares solver

While we must acknowledge that the error issue presented in Section 5.4.3 does limit the applicability of the fast QR algorithm, it is still a viable method for small M systems. One of the advantages of using the eigenfunction expansion is that it is the optimal M-term approximation to the Gaussian; therefore, at whatever point we choose to terminate (5.33) we have the best possible M-term series approximation to the observed function f. Furthermore, Table 5.1a suggests a method for terminating the iteration before error dominates the solution of $\Phi \boldsymbol{c} = \boldsymbol{y}$.

Algorithm 5 Adaptive fast least squares solver

Given $\boldsymbol{x}$, $\boldsymbol{y}$, ε, α, M
Set $c_0 = 0$
Choose an M_{tol}, to terminate when $\boldsymbol{q}_k^T \boldsymbol{y} < M_{\text{tol}}$
for $k = 1$ to M **do**
  Compute $\boldsymbol{q}_k$, σ_k and $\boldsymbol{u}_k$ using **Algorithm 4.b**
  if $\boldsymbol{q}_k^T \boldsymbol{y} > M_{\text{tol}}$ **then**
    $\boldsymbol{c}_k \leftarrow \boldsymbol{c}_{k-1} + \frac{\boldsymbol{q}_k^T \boldsymbol{y}}{\sigma_k} \boldsymbol{u}_k$
  else
    return $\boldsymbol{c}_{k-1}$
  end if
end for
return $\boldsymbol{c}_k$

This algorithm uses the magnitude of the value $\boldsymbol{q}_k^T \boldsymbol{y}$ to determine early truncation. If this inner product is ever too small, the algorithm decides that it is as accurate as possible; if the algorithm reaches the Mth iteration, then it returns the rank M least squares solution. The value M_{tol} would need to be problem dependent to make sure that the algorithm uses as many terms as possible. A general guideline might be to simply use a value no less than $M_{\text{tol}} \approx 10^{-14}$ because Figure 5.2 suggests that is roughly the minimum level of lost orthogonolization for fast QR. Using a M_{tol} less than that would open the solution to potentially significant

incorrect contributions.

5.5 Conclusions regarding recurrence relations

We have developed a technique to allow for fast least squares solves of eigenfunction approximations arising from 1D interpolation with Gaussians. This technique leverages the recurrence relation of the eigenfunctions, and was demonstrated to be asymptotically faster than the generic QR factorization.

Looking forward, it is important to continue to study the error properties of this fast QR method. Because Gaussians are expected to be spectrally accurate when recovering analytic functions, any error accumulation may be significant. At the same time, many kriging applications assume some level of noise is always present with the Gaussian process [163]. If the level of error introduced from the fast solve were less than that noise, there may be no penalty to using the fast method.

One of the great advantages of using radial basis functions for scattered data interpolation is the natural step to higher dimensions. As was seen in Section 4.3.2, the tensor product nature of the Gaussian kernel provides the eigenfunctions with a similar product structure. There is no such analogy for this fast QR method, unfortunately, which means that more analysis is needed to extend it to scattered data approximation in higher dimensions.

5.6 Future work on iterative methods for eigenfunctions

The main motivation behind the use of regression for the RBF-QRr algorithm was the growing computational cost of RBF-QR for higher dimensions. Beyond the use of $M < N$, there may be other avenues to improving the performance of RBF-QRr. One possible opportunity involves the recursive evaluation of eigenfunctions by exploiting the recursive nature of the Hermite polynomials. This section is more interested in solving the least squares system (4.21) iteratively. Here again we drop down to 1D for the analysis for simplicity.

One aspect of the low-rank approximation of the kernel comes which may be of use computationally comes from a more efficient evaluation of the interpolant. Recall that an RBF approximation $s(x)$ is evaluated via

$$s(\boldsymbol{x}) = \sum_{k=1}^{N} K(\boldsymbol{x}, \boldsymbol{x}_k) c_k$$

where $\boldsymbol{c} = \mathsf{K}^{-1}\boldsymbol{y}$ are the coefficients determined from the interpolation. Using the M-term eigenfunction expansion $K(\boldsymbol{x}, \boldsymbol{z}) \approx \sum_{j=1}^{M} \lambda_j \varphi_j(\boldsymbol{x})\varphi_j(\boldsymbol{z})$ to approximate the kernel gives

$$s(\boldsymbol{x}) \approx \sum_{k=1}^{N} \sum_{j=1}^{M} \lambda_j \varphi_j(\boldsymbol{x}) \varphi_j(\boldsymbol{x}_k) c_k.$$

We can do some rearranging in the treecode or fast multipole style to come up with a more useful way to evaluate this double summation:

$$s(\boldsymbol{x}) \approx \sum_{j=1}^{M} \varphi_j(\boldsymbol{x}) \underbrace{\lambda_j \sum_{k=1}^{N} \varphi_j(\boldsymbol{x}_k) c_k}_{\boldsymbol{b}_j}$$

$$s(\boldsymbol{x}) \approx \sum_{j=1}^{M} \varphi_j(\boldsymbol{x}) b_j$$

This means that, by precomputing the $\boldsymbol{b}$ vector from the $\boldsymbol{c}$ vector, future function evaluations $s(\boldsymbol{x})$ can be computed at cost $O(M)$.

We are in some sense already taking advantage of this when we do the regression because for RBF-QRr we are solving the system

$$\begin{aligned} \mathsf{K}\boldsymbol{c} &= \boldsymbol{y} \\ \Phi \underbrace{\Lambda \Phi^T \boldsymbol{c}}_{=\boldsymbol{b}} &\approx \boldsymbol{y} \\ \Phi \boldsymbol{b} &\approx \boldsymbol{y} \end{aligned}$$

which is a least squares problem given that $M < N$ is the number of nonzero diagonal values in Λ. What we are not taking advantage of is the encapsulation that the FMM approach provides us: the operation of evaluating the interpolant is not necessarily related to finding $\boldsymbol{b}$. Instead of choosing M equal for both determining $\boldsymbol{c}$ and evaluating $s(\boldsymbol{x})$, we could instead approach the problem in the following way

1. Choose an M_2 which will be used to evaluate $s(\boldsymbol{x})$.
2. Choose a different (likely smaller) M_1 which will be used to solve the least squares problem.

These choices are then used in an iterative solver as the matrix-vector product and preconditioner respectively.

At this point, there are several items to consider. For instance, what effect does the error from the series approximated matrix-vector product have on the iterative solve? It seems logical that, because the quality of the summation improves with M, that the matrix vector product is more accurate as M_2 increases. Perhaps of

more use is the fact that as $\varepsilon \to 0$ the eigenvalues decay more quickly, meaning that later terms in the series have less significance. Therefore, for small ε, the error in the matrix-vector product should be quite small, as should its effect. This process is similar to the Jacobian-free matrix-vector products discussed in Chapter 3, where some loss in accuracy was beneficial computationally.

We must also consider how our solution is affected by using a low-rank preconditioner for our iterative method. The system we hope to solve is rank $M_2 < N$, so we are already restricting ourselves to methods like GMRES which can handle singular systems. A likely choice of method is Minres-QLP [44], or its sibling CS-Minres, which is currently under development. In [43], suggestions were given about preconditioning inconsistent, singular linear systems, but no specific comments were made regarding the effect of a singular preconditioner.

The computational cost of each $s(\boldsymbol{x})$ evaluation in such a setting is $O(M_2)$, and since there are N points at which it needs to be evaluated, the cost of each matrix-vector product here is $O(M_2 N)$. Considering the work of finding the $\boldsymbol{b}$ terms is $O(M_2 N)$ that makes the total cost of matrix-vector multiplication $O(M_2 N)$.

The cost of preconditioning is more complicated, because we are no longer just solving the least squares problem $\Phi \boldsymbol{b} = \boldsymbol{y}$. Let's write the approximation that we are using in matrix form

$$\mathsf{K} \approx \begin{pmatrix} & & \\ \Phi_1 & \Phi_2 & \Phi_3 \\ & & \end{pmatrix} \begin{pmatrix} \Lambda_1 & & \\ & \Lambda_2 & \\ & & 0 \end{pmatrix} \begin{pmatrix} \Phi_1^T & & \\ \Phi_2^T & & \\ \Phi_3^T & & \end{pmatrix}.$$

The least squares system we need to solve for the preconditioner is

$$\Phi_1 \Lambda_1 \Phi_1^T \boldsymbol{c} = \boldsymbol{y},$$

which means that we still need the QR factorization $\mathsf{QR} = \Phi_1$ as we did for the RBF-QRr approximation. Here we need to use it both for the overdetermined system (to get $\boldsymbol{b}$) and the underdetermined system to return $\boldsymbol{c}$ for the fast matrix-vector product. After Q and R are found, we can find

$$\mathsf{QR}\Lambda_1\mathsf{R}^T\mathsf{Q}^T\boldsymbol{c} = \boldsymbol{y},$$
$$\boldsymbol{c} = \mathsf{QR}^{-T}\Lambda_1^{-1}\mathsf{R}^{-1}\mathsf{Q}^T\boldsymbol{y}.$$

The cost of finding the QR inverse is $O(NM_1^2)$, and the cost of solving for $\boldsymbol{c}$ is $O(NM_1 + N^2)$ meaning that the total cost of preconditioning is $O(NM_1^2)$. Balancing the work of preconditioning with the work of matrix-vector products would make $M_2 = M_1^2$ the logical choice, although that does not necessarily mean that a solution of rank M_1 is sufficient for accuracy purposes.

Let's see if we can simplify the work needed at all. Specifically, we might be able to leverage the fact that the structure of the matrix-vector multiplication is similar to the preconditioner. The preconditioned Krylov space we are generating for the system

$$K\boldsymbol{c} = \boldsymbol{y},$$
$$\left(\Phi_1\Lambda_1\Phi_1^T + \Phi_2\Lambda_2\Phi_2^T\right)\left(\Phi_1\Lambda_1\Phi_1^T\right)^+\left(\Phi_1\Lambda_1\Phi_1^T\right)\boldsymbol{c} \approx \boldsymbol{y}$$

is

$$\mathcal{K} = \left\{\boldsymbol{y}, \left(\Phi_1\Lambda_1\Phi_1^T + \Phi_2\Lambda_2\Phi_2^T\right)\left(\Phi_1\Lambda_1\Phi_1^T\right)^+\boldsymbol{y}, ...\right\}.$$

This system can simplify somewhat to

$$\left(\Phi_1\Lambda_1\Phi_1^T + \Phi_2\Lambda_2\Phi_2^T\right)\left(\Phi_1\Lambda_1\Phi_1^T\right)^+ = \mathsf{QQ}^T + \Phi_2\Lambda_2\Phi_2^T\mathsf{QR}^{-T}\Lambda_1^{-1}\mathsf{R}^{-1}\mathsf{Q}^T.$$

Looking at this, we can see a few important things. First of all the QQ^T term appears in each Krylov vector because it represents the solution to the M_1 least

squares system. Second the magnitude of the correction is mostly small because the values in Λ_2 are much smaller than Λ_1 and thus the product is small. This is analogous to the RBF-QR structure seen in (4.11).

If we could find a way to cheaply apply this correction term we could form the Krylov space at little cost because the $\mathsf{Q}\mathsf{Q}^T$ term can be carried over between iterations. Maybe another thought is we could first form $\Phi\mathsf{Q}\mathsf{R}^{-T}$, then analytically apply the diagonal scalings of Λ_2 and Λ_1^{-1}. $\mathsf{R}^{-1}\mathsf{Q}^T$ is the least squares solution operator, so applying that is essentially applying the preconditioner to a vector but leaving it in the M_1 space rather than bringing it back to the M_2 space.

More thought and analysis is needed to convert the existing eigenfunction framework to an approach suitable for iterative methods. Given the success of other research in performing scattered data approximation using iterative linear solvers [110], it would seem that this approach too would benefit from that possibility.

CHAPTER 6

APPROXIMATING DERIVATIVES WITH EIGENFUNCTIONS FOR MESHFREE COUPLING

6.1 Introduction

Scattered data interpolation with Gaussian kernels

$$K(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\varepsilon^2 \|\boldsymbol{x} - \boldsymbol{z}\|^2), \qquad \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^d,$$

can achieve spectral approximation rates [176], but high accuracy with few input points often occurs when the shape parameter ε produces an ill-conditioned interpolation system. This has been historically referred to as the trade-off or uncertainty principle [151] as $\varepsilon \to 0$. In Chapter 4, an eigenvalue series approximation to Gaussians in arbitrary dimensions was developed for interpolating data.

The eigenvalue decomposition method was tested on examples in 1, 2 and 5 dimensions in Chapter 4; with an appropriate choice of parameters, we were able to avoid the ill-conditioning present in the standard Gaussian basis, while still preserving the high order of accuracy associated with Gaussians. Because this technique is related to the RBF-QR method [70], we refer to it as GaussQR, or GaussQRr in the low-rank version of the algorithm.

Once higher order approximations can be conducted stably with Gaussians, derivatives of those approximations can also be computed stably. Previously, radial basis function (RBF) derivatives have appeared in boundary value problem solutions using collocation [152], and the method of fundamental solutions [55]. The first use of series expansions to stably compute derivatives appeared in [72] to solve differential equations on a sphere. RBF-FD methods involving differentiation

on a scattered grid have been presented in [181, 71] and a more general discussion of RBF derivative accuracy is found in [76].

This chapter analyzes the technique of approximating derivatives using the eigenfunction expansion, and compare the structure to the RBF-PS structure discussed in [60]. Numerical results are presented in Section 6.2.2 showing the validity of that technique for approximating various functions. Multiphysics coupling in general was discussed in Section 1.3, with the specific use of GaussQR implemented in Section 6.3. Stable derivative approximations are applied in Section 6.3.1 to an example, with improved convergence resulting from the higher order derivatives. Techniques for managing the computational cost of GaussQRr were introduced in [122], and expanded upon in Chapter 5. These computational issues, along with other coupling specific items, are addressed in Section 6.3.2. Finally, Section 6.4 describes future improvements to this method.

6.2 Approximating derivatives with eigenfunctions

The eigenfunction expansion for Gaussians in $\mathbb{R}^d$ was first discussed in [137], and further developed in Chapter 4. It was determined that

$$\exp(-\varepsilon^2\|x-z\|^2) \approx \sum_{m=1}^{M} \lambda_m \varphi_m(x)\varphi_m(z), \tag{5.1}$$

where the eigenvalues and eigenfunctions are

$$\lambda_m = \sqrt{\frac{\alpha^2}{\alpha^2+\delta^2+\varepsilon^2}}\left(\frac{\varepsilon^2}{\alpha^2+\delta^2+\varepsilon^2}\right)^{m-1}, \tag{4.4b}$$

$$\varphi_m(x) = \gamma_m \exp(-\delta^2 x^2) H_{m-1}(\beta\alpha x). \tag{4.4a}$$

Above, M is the truncation point of an otherwise infinite series, H_{m-1} is the Hermite polynomial of degree $m-1$, and α is the global scale parameter defined in

Chapter 4. The free parameter α must be chosen to uniquely define the eigenfunctions; this is analogous to how ε must be chosen to uniquely define the reproducing kernel Hilbert space spanned by the Gaussians. The other parameters are fixed to α and ε:

$$\beta = \left(1 + \left(\frac{2\varepsilon}{\alpha}\right)^2\right)^{\frac{1}{4}}, \quad \gamma_m = \sqrt{\frac{\beta}{2^{m-1}\Gamma(m)}}, \quad \delta^2 = \frac{\alpha^2}{2}\left(\beta^2 - 1\right).$$

The expansion above can be extended to multiple dimensions by exploiting the tensor product structure of the Gaussian in multiple dimensions; that was described in Chapter 4.

This section works with the eigenfunctions to establish their derivatives and express those derivatives using the recurrence relation defined in Section 5.1.2. Section 6.2.2 establishes the use of differentiation matrices which allow for the evaluation of the derivatives of interpolants computed using GaussQR and GaussQRr. These differentiation matrices are tested to determine their stability on sample problems, as well as the order of accuracy for higher derivatives.

6.2.1 Differentiating the eigenfunctions

To consider the derivatives of the eigenfunctions, we can directly differentiate (4.4a),

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\varphi_m(x) = &-2\gamma_m\delta^2 x\exp(-\delta^2x^2)H_{m-1}(\beta\alpha x)\\ &+\gamma_m\exp(-\delta^2x^2)\frac{\mathrm{d}}{\mathrm{d}x}H_{m-1}(\beta\alpha x).\end{aligned}$$

From [165], the derivative of a Hermite polynomial is

$$\frac{\mathrm{d}}{\mathrm{d}x}H_m(x) = 2mH_{m-1}(x), \qquad \frac{\mathrm{d}}{\mathrm{d}x}H_0(x) = 0.$$

Plugging this in above gives

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\varphi_m(x) =& -2\gamma_m\delta^2 x\exp(-\delta^2x^2)H_{m-1}(\beta\alpha x)\\ &+2(m-1)\beta\alpha\gamma_m\exp(-\delta^2x^2)H_{m-2}(\beta\alpha x),\end{aligned}$$

to which we can apply the identity $\sqrt{2(m-1)}\gamma_m = \gamma_{m-1}$,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\varphi_m(x) =& -2\delta^2 x\gamma_m\exp(-\delta^2x^2)H_{m-1}(\beta\alpha x)\\ &+2(m-1)\beta\alpha\frac{\gamma_{m-1}}{\sqrt{2(m-1)}}\exp(-\delta^2x^2)H_{m-2}(\beta\alpha x).\end{aligned}$$

Finally, making the substitution of (4.4a) produces

$$\frac{\mathrm{d}}{\mathrm{d}x}\varphi_m(x) = -2\delta^2\ x\ \varphi_m(x) + \sqrt{2(m-1)}\beta\alpha\ \varphi_{m-1}(x). \tag{6.1}$$

This allows us to compute the derivatives of eigenfunctions using the eigenfunctions themselves. Higher order derivatives can also be computed via direct differentiation and substitution, producing the second derivative

$$\begin{aligned}\frac{\mathrm{d}^2}{\mathrm{d}x^2}\varphi_m(x) =& 2\delta^2\ (2\delta^2x-1)\ \varphi_m(x) -\\ &4\sqrt{2}\delta^2\beta\alpha\sqrt{m-1}\ x\ \varphi_{m-1}(x) +\\ &2(\beta\alpha)^2\sqrt{(m-1)(m-2)}\ \varphi_{m-2}(x),\end{aligned}$$

the third derivative

$$\begin{aligned}\frac{\mathrm{d}^3}{\mathrm{d}x^3}\varphi_m(x) =& -4\delta^4\ x(2\delta^2x^2-3)\ \varphi_m(x) +\\ &6\sqrt{2}\delta^2\beta\alpha\sqrt{m-1}\ (2\delta^2x^2-1)\ \varphi_{m-1}(x) -\\ &12\delta^2(\beta\alpha)^2\sqrt{(m-1)(m-2)}\ x\ \varphi_{m-2}(x) +\\ &2\sqrt{2}(\beta\alpha)^3\sqrt{(m-1)(m-2)(m-3)}\ \varphi_{m-3}(x),\end{aligned}$$

and the fourth derivative

$$\begin{aligned}\frac{\mathrm{d}^4}{\mathrm{d}x^4}\varphi_m(x) =& 4\delta^4\left(4\delta^2x^4-12\delta^2x^2+3\right)\varphi_m(x) - \\ & 16\sqrt{2}\delta^4\beta\alpha\sqrt{m-1}\,x(2\delta^2x^2-3)\,\varphi_{m-1}(x) + \\ & 24\delta^2(\beta\alpha)^2\sqrt{(m-1)(m-2)}\,(2\delta^2x^2-1)\,\varphi_{m-2}(x) - \\ & 16\sqrt{2}\delta^2(\beta\alpha)^3\sqrt{(m-1)(m-2)(m-3)}\,x\,\varphi_{m-3}(x) + \\ & 4(\beta\alpha)^4\sqrt{(m-1)(m-2)(m-3)(m-4)}\,\varphi_{m-4}(x).\end{aligned}$$

As discussed in Chapter 5, the structure of the Hermite polynomials induces a three-term recurrence underlying the eigenfunctions:

$$\begin{aligned}\varphi_1(x) &= \sqrt{\beta}e^{-\delta^2x^2},\\ \varphi_2(x) &= \sqrt{2}\beta\alpha x\sqrt{\beta}e^{-\delta^2x^2},\\ \varphi_m(x) &= \sqrt{\frac{2}{m-1}}\beta\alpha x\varphi_{m-1}(x) - \sqrt{\frac{m-2}{m-1}}\varphi_{m-2}(x).\end{aligned} \tag{6.2}$$

Applying (6.2) multiple times produces the relations

$$\begin{aligned}\varphi_{m-2}(x) &= \frac{1}{\sqrt{m-2}}\sqrt{2}\beta\alpha\;x\;\varphi_{m-1}(x) - \sqrt{m-1}\;\varphi_m(x),\\ \varphi_{m-3}(x) &= \frac{2(\beta\alpha)^2x^2-m}{\sqrt{(m-2)(m-3)}}\;\varphi_{m-1}(x) - \sqrt{2}\sqrt{m-1}\beta\alpha\;x\;\varphi_m(x),\\ \varphi_{m-4}(x) &= \frac{\sqrt{2}\beta\alpha x(2(\beta\alpha)^2x^2-2m+3)}{\sqrt{(m-2)(m-3)(m-4)}}\;\varphi_{m-1}(x)\\ &\quad - \sqrt{m-1}(2(\beta\alpha)^2x^2-m+3)\;\varphi_m(x).\end{aligned}$$

This allows all the derivatives of $\varphi_m(x)$ to be written only in terms of $\varphi_m(x)$ and

$\varphi_{m-1}(x)$:

$$\begin{aligned}
\frac{\mathrm{d}^2}{\mathrm{d}x^2}\varphi_m(x) &= \Big[2\delta^2(2\delta^2x^2-1) - 2(\beta\alpha)^2(m-1)\Big]\ \varphi_m(x) + \\
&\quad 2\beta\alpha((\beta\alpha)^2 - 2\delta^2)\sqrt{2(m-1)}\ x\ \varphi_{m-1}(x), \\
\frac{\mathrm{d}^3}{\mathrm{d}x^3}\varphi_m(x) &= \Big[-8\delta^6x^3 + (4(\beta\alpha)^2(m-1)(3\delta^2 - (\beta\alpha)^2) + 12\delta^4)x\Big]\ \varphi_m(x) + \\
&\quad 2\beta\alpha\sqrt{2(m-1)}\Big[(6\delta^4 + 2(\beta\alpha)^4 - 6(\beta\alpha)^2\delta^2)x^2 - \\
&\qquad ((\beta\alpha)^2(m-2) + 3\delta^2)\Big]\ \varphi_{m-1}(x), \\
\frac{\mathrm{d}^4}{\mathrm{d}x^4}\varphi_m(x) &= 4\Big[2(4\delta^2(\beta\alpha)^4(m-1) - 6\delta^2(\beta\alpha)^2(m-1) - (\beta\alpha)^6(m-1) - 6\delta^6)x^2 + \\
&\qquad 4\delta^8x^4 + (\beta\alpha)^4(m-3)(m-1) + 3\delta^4 + 6\delta^2(\beta\alpha)^2(m-1)\Big]\ \varphi_m(x) + \\
&\quad 4\beta\alpha\sqrt{2(m-1)}\Big[2((\beta\alpha)^6 - 4\delta^2(\beta\alpha)^4 + 6\delta^4(\beta\alpha)^2 - 4\delta^6)x^2 + \\
&\qquad 2(\beta\alpha)^2(m-1)(2\delta^2 - (\beta\alpha)^2) + 12\delta^4 + \\
&\qquad 3(\beta\alpha)^4 - 10\delta^2(\beta\alpha)^2\Big]\ x\ \varphi_{m-1}(x).
\end{aligned}$$

We list here only the first four derivatives because those are the only ones considered in this thesis. Higher derivatives can be determined following the same pattern as above.

6.2.2 Using differentiation matrices on interpolants

Now that we have the ability to differentiate eigenfunctions in one dimension, we need to determine how to evaluate the derivative of multidimensional interpolants produced using GaussQR. Traditional kernel interpolation in $\mathbb{R}^d$ takes N function values $(y_n)_{n=1}^N$ evaluated at N nodes $(\boldsymbol{x}_n)_{n=1}^N$ where $\boldsymbol{x}_n \in \mathbb{R}^d$ and produces an approximation $s(\boldsymbol{x})$ to the unknown function $y(\boldsymbol{x})$. The interpolant takes the

form

$$s(\boldsymbol{x}) = \sum_{n=1}^{N} c_n K(\boldsymbol{x}, \boldsymbol{x}_n),$$

where the $\boldsymbol{c}$ values are determined by solving the interpolation equations

$$\mathsf{K}\boldsymbol{c} = \boldsymbol{y}. \tag{4.14}$$

The matrix $(\mathsf{K})_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is symmetric positive definite when K is the Gaussian, and $(\boldsymbol{y})_i = y_i$ and $(\boldsymbol{c})_i = c_i$ as should be expected. We can write $s(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T\boldsymbol{c}$, where

$$\boldsymbol{k}(\boldsymbol{x})^T = (K(\boldsymbol{x}, \boldsymbol{x}_1), \cdots, K(\boldsymbol{x}, \boldsymbol{x}_N)).$$

Replacing the kernel matrix K by the eigenfunction expansion using (5.1) produces

$$\mathsf{K} \approx \Phi\Lambda\Phi^T.$$

Here, $\Phi \in \mathbb{R}^{N\times M}$ has ith row $(\Phi)_{i,:} = \boldsymbol{\phi}(\boldsymbol{x}_i)^T$ where

$$\boldsymbol{\phi}(\boldsymbol{x})^T = (\varphi_{\boldsymbol{m}_1}(\boldsymbol{x}) \quad \cdots \quad \varphi_{\boldsymbol{m}_M}(\boldsymbol{x})),$$

and $\Lambda \in \mathbb{R}^{M\times M}$ is a diagonal matrix with ith diagonal value $\lambda_{\boldsymbol{m}_i}$. The ith multi-index is $\boldsymbol{m}_i$, which has d values and describes the order of the components of the ith eigenfunction,

$$\varphi_{\boldsymbol{m}_i}(\boldsymbol{x}) = \prod_{k=1}^{d} \varphi_{(\boldsymbol{m}_i)_k}(x_k). \tag{6.3}$$

This was first introduced in Section 4.3.2. As an example, $\varphi_{[3,5]}(x_1, x_2) = \varphi_3(x_1)\varphi_5(x_2)$.

It is necessary to consider two cases for the length of the eigenfunction expansion M: $M < N$ or $M \geq N$. If we choose $M < N$, which is more likely when N is

large, the low rank approximation system

$$\Phi \boldsymbol{b} = \boldsymbol{y} \tag{6.4}$$

replaces (4.14). We refer to this low rank regression method as GaussQRr, to correspond to the RBF-QRr terminology defined in Chapter 4. The solution $\boldsymbol{b} \in \mathbb{R}^M$ can be thought of as $\Lambda \Phi^T \boldsymbol{c}$, although not computed that way because of the danger of computing $\boldsymbol{c}$. In this formulation, $s(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{b}$.

Differentiating $s(\boldsymbol{x})$ is straightforward because the only dependence on $\boldsymbol{x}$ appears in the eigenfunctions. Using $\mathcal{D}_k$ to indicate the derivative with respect to the kth dimension,

$$\begin{aligned} \mathcal{D}_k s(\boldsymbol{x}) &= \mathcal{D}_k \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{b}, \\ \mathcal{D}_k s(\boldsymbol{x}) &= (\mathcal{D}_k \varphi_{\boldsymbol{m}_1}(\boldsymbol{x}) \quad \cdots \quad \mathcal{D}_k \varphi_{\boldsymbol{m}_M}(\boldsymbol{x})) \boldsymbol{b}. \end{aligned} \tag{6.5}$$

Since multiple dimension eigenfunctions are formed by the tensor product structure in (6.3), the derivative in one dimension does not affect the others, thus

$$\mathcal{D}_k \varphi_{\boldsymbol{m}}(\boldsymbol{x}) = \varphi'_{(\boldsymbol{m})_k}(x_k) \prod_{\substack{j=1 \\ j \neq k}}^{d} \varphi_{(\boldsymbol{m})_j}(x_j). \tag{6.6}$$

For some applications, it is useful to describe this differentiation process using the so-called *differentiation matrix* framework [168, 60]. If we define the matrix $(\Phi^{\mathcal{D}_k})_{i,:} = \mathcal{D}_k \boldsymbol{\phi}(\boldsymbol{x}_i)^T$ then the relationship

$$\mathcal{D}_k \begin{pmatrix} s(\boldsymbol{x}_1) \\ \vdots \\ s(\boldsymbol{x}_N) \end{pmatrix} = \Phi^{\mathcal{D}_k} \Phi^{+} \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

must hold, and $\Phi^{\mathcal{D}_k} \Phi^{+}$ would be the differentiation matrix in the kth dimension. Because $M < N$, Φ^{-1} does not exist, and (6.4) must be solved with the pseudoinverse [84], denoted Φ^{+}.

The term differentiation matrix is appropriate because s is built to approximate y, so the matrix $\Phi^{\mathcal{D}_k}\Phi^+$ is accepting function values and returning values of the derivative of the function evaluated at those points. Note that $\Phi^{\mathcal{D}_k}\Phi^+$ is an outer product, and thus has rank M. Combining (6.4) and (6.5) gives a similar structure for the derivative at any point $\boldsymbol{x}$ given $\boldsymbol{y}$,

$$\mathcal{D}_k s(\boldsymbol{x}) = \mathcal{D}_k \boldsymbol{\phi}(\boldsymbol{x})^T \Phi^+ \boldsymbol{y}.$$

More generally, the derivative evaluated at $\hat{N}$ points $(\hat{\boldsymbol{x}}_n)_{n=1}^{\hat{N}}$ can be written as

$$\begin{aligned}\mathcal{D}_k \begin{pmatrix} s(\hat{\boldsymbol{x}}_1) \\ \vdots \\ s(\hat{\boldsymbol{x}}_{\hat{N}}) \end{pmatrix} &= \begin{pmatrix} \mathcal{D}_k \boldsymbol{\phi}(\hat{\boldsymbol{x}}_1) \\ \vdots \\ \mathcal{D}_k \boldsymbol{\phi}(\hat{\boldsymbol{x}}_{\hat{N}}) \end{pmatrix} \Phi^+ \boldsymbol{y} \\ &= \hat{\Phi}^{\mathcal{D}_k} \Phi^+ \boldsymbol{y} \end{aligned} \tag{6.7}$$

where $\hat{\Phi}^{\mathcal{D}_k}\Phi^+$ is the differentiation matrix which accepts function values at $\chi_{\text{IN}} = (\boldsymbol{x}_n)_{n=1}^N$ and returns derivatives at $\chi_{\text{OUT}} = (\hat{\boldsymbol{x}}_n)_{n=1}^{\hat{N}}$.

When $M \geq N$ we use the GaussQR formulation derived in Chapter 4: the Gaussian basis

$$\begin{aligned}\boldsymbol{k}(\boldsymbol{x})^T &= \begin{pmatrix} K(\boldsymbol{x}, \boldsymbol{x}_1) & \cdots & K(\boldsymbol{x}, \boldsymbol{x}_N) \end{pmatrix} \\ &= \begin{pmatrix} \varphi_{\boldsymbol{m}_1}(\boldsymbol{x}) & \cdots & \varphi_{\boldsymbol{m}_M}(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \Phi_2^T \Phi_1^{-T} \Lambda_1^{-1} \end{pmatrix} \Lambda_1 \Phi_1^T \end{aligned}$$

is converted to the stable basis

$$\begin{aligned}\boldsymbol{\psi}(\boldsymbol{x})^T &= \begin{pmatrix} \psi_1(\boldsymbol{x}) & \cdots & \psi_N(\boldsymbol{x}) \end{pmatrix} \\ &= \begin{pmatrix} \varphi_{\boldsymbol{m}_1}(\boldsymbol{x}) & \cdots & \varphi_{\boldsymbol{m}_M}(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2 \Phi_2^T \Phi_1^{-T} \Lambda_1^{-1} \end{pmatrix}, \end{aligned}$$

where $\Phi = (\Phi_1\ \Phi_2)$ is defined, as before, by $(\Phi)_{i,j} = \varphi_{\boldsymbol{m}_j}(\boldsymbol{x}_i)$. The submatrices composing Φ have size $\Phi_1 \in \mathbb{R}^{N\times N}$ and $\Phi_2 \in \mathbb{R}^{N\times M-N}$, and $(\Lambda)_{i,i} = \lambda_{\boldsymbol{m}_i}$ has diagonal blocks $\Lambda_1 \in \mathbb{R}^{N\times N}$ and $\Lambda_2 \in \mathbb{R}^{M-N\times M-N}$. In terms of this new basis, the linear system

$$\Psi\hat{\boldsymbol{b}} = \boldsymbol{y}$$

replaces (4.14) in defining the interpolation, where $(\Psi)_{i,:} = \boldsymbol{\psi}(\boldsymbol{x}_i)^T$, and now $\hat{s}(\boldsymbol{x}) = \boldsymbol{\psi}(\boldsymbol{x})^T\hat{\boldsymbol{b}}$. We can now differentiate directly to see that derivatives of the stable basis are expressed in terms of derivatives of the eigenfunctions:

$$\mathcal{D}_k\boldsymbol{\psi}(\boldsymbol{x})^T = \mathcal{D}_k\boldsymbol{\phi}(\boldsymbol{x})^T \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2\Phi_2^T\Phi_1^{-T}\Lambda_1^{-1} \end{pmatrix}. \tag{6.8}$$

The derivative of the eigenfunctions was discussed separately for the case $M < N$ above. Derivatives of the approximation at any point can be computed with

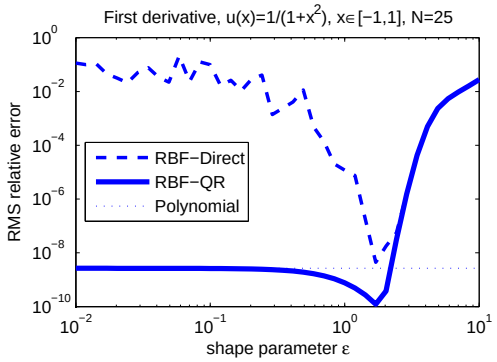
$$\begin{aligned}\mathcal{D}_k s(\boldsymbol{x}) &= \mathcal{D}_k\boldsymbol{\psi}(\boldsymbol{x})^T\Psi^{-1}\boldsymbol{y} \\ &= \mathcal{D}_k\boldsymbol{\phi}(\boldsymbol{x})^T \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2\Phi_2^T\Phi_1^{-T}\Lambda_1^{-1} \end{pmatrix} \Psi^{-1}\boldsymbol{y},\end{aligned}$$

allowing for the same differentiation matrix structure to form as before.

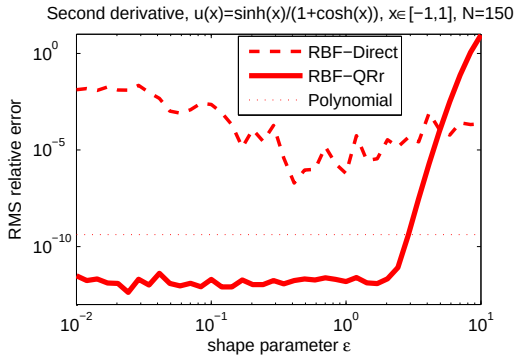
6.2.3 Numerical results

We consider now some different results involving approximating derivatives using the eigenfunction expansion. In the first instance, we study first and second order derivatives computed using (6.5) and (6.8), displayed in Figure 6.1.

These tests show that the eigenfunction expansion provides the same potential for accurate derivative approximation as it did for function recovery in Chapter



(a) The full series derivative with GaussQR using (6.8). An optimal RBF interpolant is more accurate than the polynomial.



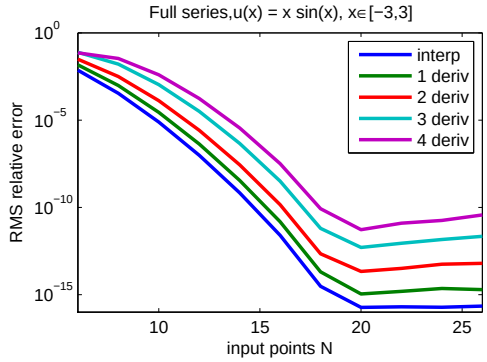
(b) The low-rank series ($M = .7N$) second derivative with GaussQRr using (6.5). RBF approximation provides better accuracy than the polynomial.

Figure 6.1: Sample points come from the Chebyshev nodes, and error is tested on those nodes. For these tests, GaussQR uses $\alpha = 3$.

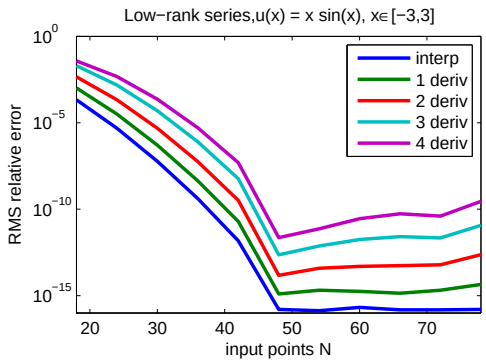
4. In Figure 6.1a we can see that the first derivative is approximated successfully with a standard $\varepsilon \to 0$ curve. It seems that the RBF interpolant reaches its asymptotic bound near $\varepsilon \approx .1$; the asymptotic limit is the polynomial interpolant [21], generated with code from [168].

For Figure 6.1b we consider a problem where the number of points is much greater, $N = 150$, encouraging the use of low rank approximation. Even in this case, there is still greater accuracy for the GaussQRr method when ε is small. The size of N here is too great for RBF-Direct to converge reliably for the ε range considered, and the low-rank eigenfunction expansion is inaccurate for larger ε, so none of these RBF methods are appropriate for large ε.

Figure 6.2 displays the accuracy for multiple derivatives as a function of the number of input points. As in the last example, the input points were placed on the Chebyshev nodes. The series approximation is able to recover the correct derivatives accurately up until machine precision dominates the accuracy. See [176] or [60] for further information regarding order of convergence for the derivatives of



(a) The full series derivative with GaussQR using (6.8). Roundoff limits accuracy beyond $N \approx 20$.



(b) The low-rank series ($M = .4N$) with GaussQRr using (6.5). Roundoff limits accuracy beyond $N \approx 50$.

Figure 6.2: Sample points come from the Chebyshev nodes, and error is tested on 200 evenly spaced points. For these tests, GaussQR uses $\varepsilon = .1$ and $\alpha = 2$.

RBF interpolants. Analysis of series approximation accuracy for a class of kernels near Gaussians can be found in [185].

6.3 Meshfree coupling

In Section 1.3, the interface conditions (1.5) were introduced in a continuous setting; actually performing the simulation requires discretizing the interface conditions. As mentioned earlier, there are two choices when enforcing the interface conditions: a specific discretization can be chosen, the required nodes of which form the resulting coupling region; or a coupling region can be fixed, and the discretization is designed to best accommodate that choice. Because the interior is the dominant portion of the simulation, the discretization is generally chosen in deference to the needs of the PDE, not the interface conditions.

Typically, the discretization scheme on the interface is the same as the discretization on the interior and coupling regions. This is a logical choice because the placement of the nodes in the coupling region is designed around the interior

discretization, and using the same discretization on the interface should take advantage of the placement of the coupling nodes. In spite of the common sense nature of this approach, there is nothing to guarantee that the best approximation to the interface conditions is achieved by using the interior discretization. This section presents an example using finite differences (FD) on the interior and coupling regions, and compare the results of a finite difference interface discretization to an RBF interface discretization.

The use of a meshfree (MF) approximation scheme in the interface region of PDEs discretized by other means was discussed in [3], [177] and [178]. The improved approximation and stability results using GaussQR Chapter 4 provides an opportunity to build on the already existing meshfree coupling framework from Wendland.

6.3.1 Example - coupled 2D heat equation

As an example of the usefulness of this meshfree coupling method, we consider in this section a linear parabolic boundary value problem

$$u_t - (u_{xx} + u_{yy}) = e^{-t}\left(1 + \frac{1}{2}(x^2 + y^2)\right), \qquad (x, y) \in \Omega, \tag{6.9a}$$

$$u = e^{-t}\left(1 - \frac{1}{2}(x^2 + y^2)\right), \qquad (x, y) \in \partial\Omega, \tag{6.9b}$$

$$u = 1 - \frac{1}{2}(x^2 + y^2), \qquad t = 0. \tag{6.9c}$$

This is a 2D version of the linear critical gradient model, introduced in Section 3.4; for simplicity, the diffusivity κ from (3.6) is set identically to 1. Ω is the full domain of the coupled problem, such that $\Omega = \Omega_1 \cup \Omega_2$. For Model 1,

$$\Omega_1 = \{(x, y) : -1 \leq x \leq 0,\ 0 \leq y \leq 1\},$$

and for Model 2,

$$\Omega_2 = \{(x, y) : 0 \leq x \leq 1,\ 0 \leq y \leq 1\}.$$

Fourth-order finite differences are used to discretize in space, and the backward Euler method is used to discretize in time; the vector $\boldsymbol{u}^{(k)}$ represents the computed solution at time $t_k = k\Delta t$. After discretizing, we are left with the "nonlinear" system

$$F(\boldsymbol{u}^{(k)}) = \frac{1}{\Delta t}(\boldsymbol{u}^{(k)} - \boldsymbol{u}^{(k-1)}) - \mathsf{L}_{FD,BC}(\boldsymbol{u}^{(k)}) - \boldsymbol{f} = 0,$$

where $\mathsf{L}_{FD,BC}$ is the Laplacian operator or identity depending on whether (6.9a) or (6.9b), respectively, is applicable. The vector $\boldsymbol{f}$ is the right hand side associated with either (6.9a) or (6.9b), and $\boldsymbol{u}^{(0)}$ is generated by (6.9c). Although the discussion earlier focused on solving nonlinear systems, the discussion is still applicable here because this linear system is just a special case of a nonlinear system.

Because the boundary condition (6.9b) is only defined on $\partial\Omega$, there must be a second order interface condition on the shared boundary between for the problem to be well-posed. Fundamentally, this is because each model needs a simulated boundary condition to allow it to be well-posed as an independent problem; once it has a "boundary condition" on the interface, it inherits the well-posedness properties associated with any single domain boundary value problem. As mentioned earlier, guaranteeing accuracy, either *a priori* or *a posteriori*, for the coupled problem is still an open problem and beyond the scope of this work.

The choice of second order condition we make here is to require values and normal derivatives of u to be equal at the shared interface of the two models. By defining u_1 and u_2 as the solutions for models 1 and 2 respectively, and $\Omega_I =$

$\Omega_1 \cap \Omega_2$, we can write the interface conditions as

$$\left.\begin{aligned} u_1 &= u_2 \\ \tfrac{\partial}{\partial x} u_1 &= \tfrac{\partial}{\partial x} u_2 \end{aligned}\right\}, \qquad (x, y) \in \Omega_I.$$

It is noteworthy that these are not the only acceptable choice of interface conditions. Any choice which produces one unique manufactured "boundary condition" per model would be acceptable. Choosing Dirichlet and Robin interface conditions would be acceptable, but it would not be acceptable to use Dirichlet to match solution values, and then have a second conditions which matched twice the values. Doing so would produce a Jacobian of the form (1.6) with fourth and sixth block rows equivalent except for a factor of two, and such a system is underdetermined.

We choose to use the Dirichlet coupling condition as F_1^I and the Neumann coupling condition as F_2^I, although this choice is arbitrary and could just as easily have been the opposite. Discretizing these interface conditions using the low rank series approximation (6.4) converts the general Jacobian matrix (1.6) to the matrix

$$\begin{pmatrix} \mathsf{J}_1(F_1) & & \mathsf{J}_1^C(F_1) & & & \\ & \mathsf{J}_2(F_2) & & & \mathsf{J}_2^C(F_2) & \\ \mathsf{J}_1(F_1^C) & & \mathsf{J}_1^C(F_1^C) & \mathsf{J}_1^I(F_1^C) & & \\ & & & \mathsf{I} & [\ \Phi_{2\to 1}\Phi_2^+\ ] & \\ & \mathsf{J}_2(F_2^C) & & & \mathsf{J}_2^C(F_2^C) & \mathsf{J}_2^I(F_2^C) \\ & & [\ \Phi_{1\to 2}^{\mathcal{D}_x}\Phi_1^+\ ] & & [\ \Phi_2^{\mathcal{D}_x}\Phi_2^+\ ] & \end{pmatrix}, \qquad \boldsymbol{u} = \begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{u}_1^C \\ \boldsymbol{u}_1^I \\ \boldsymbol{u}_2^C \\ \boldsymbol{u}_2^I \end{pmatrix}.$$

Note the inclusion of the solution vector to the right, for reference. This matrix has mostly the same structure and content, because most of the entries are not determined by the interface conditions. Two block rows have been converted in this matrix in order to satisfy the interface conditions:

Fourth block row This row defines F_1^I as the Dirichlet condition, thus requiring

that solution values from model 2 be mapped to the mesh of model 1.

- $\begin{pmatrix} \mathsf{J}_1^C(F_1^I) & \mathsf{J}_1^I(F_1^I) \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & \mathsf{I} \end{pmatrix}$. Because the residual F_1^I is being evaluated on the interface nodes of model 1, $\boldsymbol{x}_1^I$, we need to produce the solution $\boldsymbol{u}_1^I$. Doing so requires only the identity I located as seen above.
- $\begin{pmatrix} \mathsf{J}_2^C(F_1^I) & \mathsf{J}_2^I(F_1^I) \end{pmatrix} \longrightarrow \begin{bmatrix} \Phi_{2\to 1}\Phi_2^+ \end{bmatrix}$. Here we are given values on $\boldsymbol{x}_2^C$ and $\boldsymbol{x}_2^I$ and asked to produce values on $\boldsymbol{x}_1^I$. The matrix $\Phi_{2\to 1}\Phi_2^+$ is like the differentiation matrix defined in (6.7), where the differential operator is replaced by the identity. Using that design, $\chi_{\text{IN}} = \{\boldsymbol{x}_2^C, \boldsymbol{x}_2^I\}$, $\chi_{\text{OUT}} = \boldsymbol{x}_1^I$ and $\boldsymbol{y} = \begin{pmatrix} \boldsymbol{u}_2^C \\ \boldsymbol{u}_2^I \end{pmatrix}$.

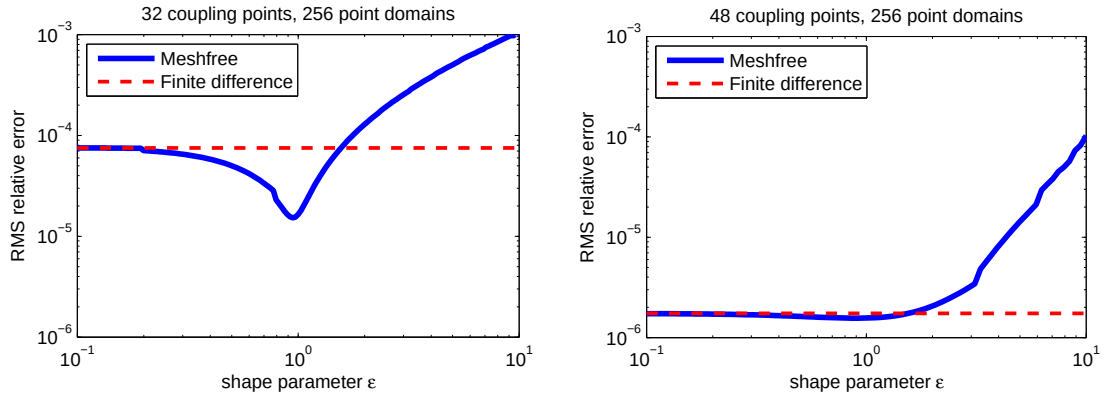
Sixth block row This row defines F_2^I as the Neumann condition, thus requiring that derivative values from model 1 be mapped to the mesh of model 2.

- $\begin{pmatrix} \mathsf{J}_1^C(F_2^I) & \mathsf{J}_1^I(F_2^I) \end{pmatrix} \longrightarrow \begin{bmatrix} \Phi_{1\to 2}^{\mathcal{D}_x}\Phi_1^+ \end{bmatrix}$. This follows directly from the differentiation matrix formula (6.7) using $\chi_{\text{IN}} = \{\boldsymbol{x}_1^C, \boldsymbol{x}_1^I\}$, $\chi_{\text{OUT}} = \boldsymbol{x}_2^I$, $\boldsymbol{y} = \begin{pmatrix} \boldsymbol{u}_1^C \\ \boldsymbol{u}_1^I \end{pmatrix}$ and $\mathcal{D}_x$ is the derivative in the x direction (normal to the interface).
- $\begin{pmatrix} \mathsf{J}_2^C(F_2^I) & \mathsf{J}_2^I(F_2^I) \end{pmatrix} \longrightarrow \begin{bmatrix} \Phi_2^{\mathcal{D}_x}\Phi_2^+ \end{bmatrix}$. Following again from the differentiation matrix formula with $\chi_{\text{IN}} = \{\boldsymbol{x}_2^C, \boldsymbol{x}_2^I\}$, $\chi_{\text{OUT}} = \boldsymbol{x}_1^I$ and $\boldsymbol{y} = \begin{pmatrix} \boldsymbol{u}_2^C \\ \boldsymbol{u}_2^I \end{pmatrix}$ and $\mathcal{D}_x$ is the derivative in the x direction (normal to the interface).

We now consider an example where the simulation is run for a single time step of size $\Delta t = .01$ and $N = 256$ uniformly distributed points in each domain. The solution technique for each time step involves using Jacobian-free Newton-Krylov,

with split finite differencing as described in Chapter 3. Two finite differencing parameters were chosen, with the largest half of the vector differenced using 10^{-7} and the smallest half differenced using 10^{-5}.

The preconditioner is FieldSplit style (recall Chapter 2), with an approximate Jacobian partially computed through coloring. Interior regions are treated with algebraic multigrid, and the remaining fields use ILU(0). The only exception is the interface regions computed with GaussQRr, since those regions are fully dense and would not benefit from ILU(0). Instead, they are preconditioned with the ILU(0) decomposition of the FD interface blocks, for reasons described below and in Section 6.3.2. Figure 6.3 compares the error present in the fully coupled system when the interface conditions are discretized using finite differences and low rank Gaussians.



(a) Coupling width Δx

(b) Coupling width $2\Delta x$

Figure 6.3: There is a range of ε values for which the meshfree approximation produces accuracy that can not be attained by finite differences. In the plot to the left, there is a pronounced benefit to using RBFs, whereas the plot to the right sees little gain because the coupling error is now on level with the interior error. For these tests, GaussQRr uses the parameters $\alpha = 1$, $M = 24$.

In Figure 6.3a, the coupling region has width 1, which is to say that one vertical strip of points adjacent to the interface region is used to approximate the derivatives. Another description of this set is the set of points with distance Δx

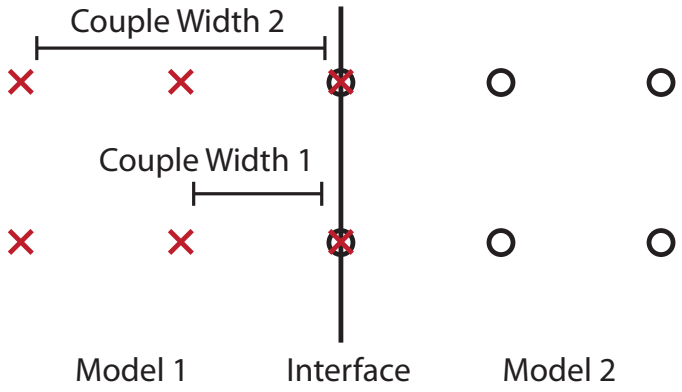


Figure 6.4: The order of the finite difference stencil determines the thickness of the coupling region. To fairly compare the FD approach to the GaussQRr approach, the same coupling width is considered for both methods.

to the interface. Figure 6.3b shows results with coupling width 2. To understand the difference between coupling width 1 and coupling width 2, see Figure 6.4. The standard one-sided finite difference approximations

$$\begin{aligned} &\text{Couple width 1:} && \frac{\partial}{\partial x}u(x) = \frac{u(x+\Delta x) - u(x)}{\Delta x} \\ &\text{Couple width 2:} && \frac{\partial}{\partial x}u(x) = \frac{u(x+2\Delta x) - 4u(x+\Delta x) + 3u(x)}{2\Delta x} \end{aligned}$$

derived from the Taylor series are used here because the interface nodes of both models are aligned. For mismatched grids we would need a more complicated finite difference approximation.

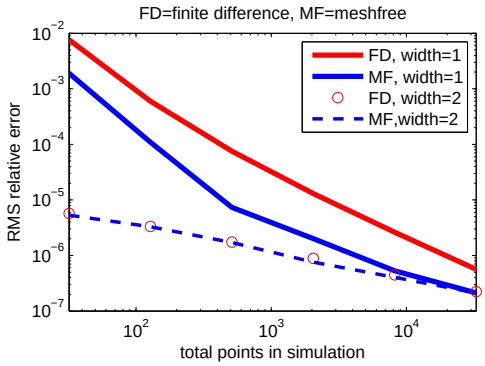
Because of the relationship between finite difference approximation and Taylor series, the FD expanded at each point in $\boldsymbol{u}^I$ is bounded by the accuracy which can be obtained by a polynomial in the x direction. The RBF interpolant has the free parameter ε, which in the limit $\varepsilon \to 0$ reproduces a polynomial, leaving the potential to see better accuracy than the FD coupling. As a reference, for larger ε, the Gaussians become unacceptably localized and produce inaccurate interpolants. Decreasing ε produces wider Gaussians which allows more interaction to produce more accurate interpolants, until the Runge phenomenom error emerges. This

source of error for very large and very small ε may result in some intermediate value where accuracy is optimal [75]. Such is the case in Figure 6.3a, where, when using the same coupling region, the meshfree interpolant produced 4 times more accurate results for $\varepsilon \approx 1$.

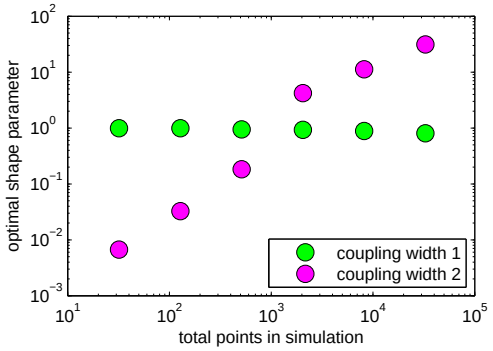
This improvement in accuracy is also related to the idea that using values in all directions around a point better approximates the derivative at that point, as discussed in [76]. Furthermore, we see that the Gaussian interpolant is approaching the finite difference interpolant as $\varepsilon \to 0$; this is expected because the FD method is a polynomial expansion, and Gaussians reproduce polynomials for $\varepsilon \to 0$. This behavior motivated the choice of the FD interface Jacobian components as the preconditioner for the dense, low-rank block of the GaussQRr interface Jacobian components.

For the finite difference coupling to catch up in accuracy to the meshfree coupling, more points must be included in the derivative computation. By increasing the coupling width to $2\Delta x$ (i.e., the two strips of points nearest to the interface) we see in Figure 6.3b that the optimal coupling strategy and the finite difference approach are very close.

Figure 6.5a shows the convergence of the coupling scheme as the number of points in the fully coupled simulation is increased. Here we can see that the accuracy of the meshfree (MF) approach is consistently better than the finite difference (FD) approach for the thinner coupling region. At $N = 32768$ we see that the accuracy of the meshfree coupling for Δx thickness is actually comparable to the $2\Delta x$ thickness case. This means that for discretizations of that size or greater, the GaussQRr coupling allows for the same accuracy while involving fewer points in the coupling.



(a) The choice of coupling strategy may play a large role in the accuracy. When fewer points are considered in the coupling region, the meshfree approximation outperforms finite differences. For a large enough simulation, the meshfree scheme can recover the coupling width=$2\Delta x$ solution with only coupling width=Δx points.



(b) For the thin coupling region, ε stays relatively constant, meaning that more points are considered in computing the interpolant. With a thick coupling region, more information is provided in the x direction, and the RBF interpolant chooses a larger ε to emphasize those points over points further away in the y direction.

Figure 6.5: We consider the effect of the coupling strategy given varying values of N, the number of points in the simulation. For these tests, GaussQRr uses the parameters $\alpha = 1$, $M = .5N$.

Earlier in Figure 6.3 we showed that there was a strong dependence on the value of ε in producing an optimal approximation for the meshfree coupling approach. In Figure 6.5b we can see that the optimal value of ε changes very little for the thin coupling case. Quite the opposite is true when the coupling width is set to $2\Delta x$, as the optimal ε increases with increases in M.

This growing ε limits the effect of points further away, which tells us that when given more data in the x direction, data in the y direction becomes less valuable. When that data is unavailable, as is the case for the thin coupling strip, the optimal interpolant continues to consider all the points available to it.

6.3.2 Computational considerations

There are some practical concerns about the viability of GaussQRr as a multi-physics coupling methods. One such concern regards the use of Newton's method,

$$\boldsymbol{u}^k = \boldsymbol{u}^{k-1} - \mathsf{J}(F)(\boldsymbol{u}^{k-1})^{-1}F(\boldsymbol{u}^{k-1}), \qquad \boldsymbol{u}^0 \text{ given},$$

to solve the system $F(\boldsymbol{u}) = 0$, where $\boldsymbol{u}^k$ is the kth Newton iteration; note that this notation is now overriding the previous notation where $\boldsymbol{u}^{(k)}$ was the kth time step. Newton's method suffers from several logistical barriers including evaluating the Jacobian.

As described, we use Jacobian-free Newton-Krylov to solve $F(\boldsymbol{u}) = 0$. This allows for matrix-vector products $\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b}$ to be approximated using the finite difference formula

$$\mathsf{J}(F)(\boldsymbol{u})\boldsymbol{b} \approx \frac{1}{h}(F(\boldsymbol{u} + h\boldsymbol{b}) - F(\boldsymbol{u})), \tag{6.10}$$

as explained in Section 1.2. Using (6.10) is appropriate when computing the matrix vector components involving the PDE and boundary conditions because they may be nonlinear (although not in the example above), but the discretization of the interface through GaussQRr is linear. The rows associated with F_1^I and F_2^I can be computed using only matrix vector products involving eigenfunctions, and that should be exploited.

Beyond simply using the available matrices to compute $\mathsf{J}(F_1^I)(\boldsymbol{u})\boldsymbol{b}$ and $\mathsf{J}(F_2^I)(\boldsymbol{u})\boldsymbol{b}$ rather than (6.10), it is also useful to note the structure of the differentiation matrices. In Section 6.2.2 it was mentioned that differentiation matrices of the form $\hat{\boldsymbol{\Phi}}^{\mathcal{D}_k}\boldsymbol{\Phi}^+$ have an outer product structure. Because of this it is much more efficient to evaluate $\hat{\boldsymbol{\Phi}}^{\mathcal{D}_k}(\boldsymbol{\Phi}^+\boldsymbol{b})$ rather than form the differentiation matrix and then

conduct the matrix-vector product $(\hat{\Phi}^{\mathcal{D}_k}\Phi^+)\boldsymbol{b}$. Although the results in Figure 6.5a were produced using the fixed proportion $M = .5N$, it may be possible to reduce M and still maintain a high order of accuracy. Doing so would save on computational cost for Jacobian-vector products, but finding the minimum acceptable M value to approximate derivatives would depend on many factors; this problem is addressed briefly in Chapter 8.

One option which allows us to use a smaller M is by choosing eigenfunctions which take advantage of the location of the available coupling points. All the previous computations used eigenfunctions of limited order in the x direction; the first 12 eigenfunction indices are listed in Table 6.1.

	$\boldsymbol{m}_1$	$\boldsymbol{m}_2$	$\boldsymbol{m}_3$	$\boldsymbol{m}_4$	$\boldsymbol{m}_5$	$\boldsymbol{m}_6$	$\boldsymbol{m}_7$	$\boldsymbol{m}_8$	$\boldsymbol{m}_9$	$\boldsymbol{m}_{10}$	$\boldsymbol{m}_{11}$	$\boldsymbol{m}_{12}$
x order	1	2	1	2	1	2	1	2	1	2	1	2
y order	1	1	2	2	3	3	4	4	5	5	6	6

(a) Couple width 1

	$\boldsymbol{m}_1$	$\boldsymbol{m}_2$	$\boldsymbol{m}_3$	$\boldsymbol{m}_4$	$\boldsymbol{m}_5$	$\boldsymbol{m}_6$	$\boldsymbol{m}_7$	$\boldsymbol{m}_8$	$\boldsymbol{m}_9$	$\boldsymbol{m}_{10}$	$\boldsymbol{m}_{11}$	$\boldsymbol{m}_{12}$
x order	1	2	1	3	2	1	3	2	1	3	2	1
y order	1	1	2	1	2	3	2	3	4	3	4	5

(b) Couple width 2

Table 6.1: The lower coupling width simulations choose the lower order eigenfunctions in the x direction.

See Section 4.4.2 for an introduction to the arrangement of these eigenfunctions. These eigenfunctions are chosen so that the maximum order in the x dimension is one more than the couple width: this was a logical choice because in the polynomial limit $\varepsilon \to 0$ the highest unique polynomial that could be fit to those points is of degree couple width plus 1. Adding more complexity in the x direction may end up overfitting the data, and adding more unnecessary computation. This is less the case on an unstructured grid, but in circumstances where the coupling region is strongly anisotropic (one dimension has much greater width than the others)

this technique helps balance contributions in all dimensions.

Now that we have considered matrix-vector products, we must also consider preconditioning, a fundamental part of any iterative linear solver and necessary to find acceptable search directions for Jacobian-free Newton-Krylov. Fortunately, the work here has already been done. Whatever technique would be normally used to precondition the FD linear system could also be used to precondition the MF system. This should be an effective preconditioning technique because the two matrices have the same effect when applied to the solution vector. For the results in Section 6.3.1, the incomplete LU factorization of the FD system was used as the preconditioner of the GMRES iterations of both the FD and MF coupling.

Another significant advantage to using RBF methods to perform the coupling is that these methods work equally well in arbitrary dimensions and on arbitrary node distributions. The RBF differentiation matrix computation (6.7) is carried out similarly regardless of the point distribution, whereas new finite difference relations would need to be computed at each point on scattered data. Furthermore, using the technique of producing multi-dimensional, higher order finite difference schemes by annihilating certain polynomials fails on many point sets in a scattered data setting. The advantages of RBF methods are described in [71, 181, 67].

6.4 Summary

We have reviewed the structure of Gaussian eigenfunction series as derived in Chapter 4, and found the derivatives of those eigenfunctions. Using the underlying three-term recurrence, we found a method to define higher derivatives of $\varphi_m(x)$ in terms of only $\varphi_m(x)$ and $\varphi_{m-1}(x)$. These derivative formulae allow us to compute

the derivative of interpolants s generated using the eigenfunction basis. Numerical results were presented confirming the quality of the derivative approximation when computed using ε values typically associated with ill-conditioning.

After considering stable derivative approximation with Gaussians, we discussed multiphysics coupling via interpolation as an application which may benefit from this technique of derivative approximation. Because RBF interpolants can be more accurate than polynomial interpolants, we attempted to use a Gaussian interpolant to discretized the interface conditions, introduced in Section 1.3, while the remaining portion of the PDE domain was discretized by other means. The results in Section 6.3.1 showed that this RBF method is capable of producing a more accurate simulation.

Most of the Jacobian was able to retain its structure because the interface conditions govern only a small portion of the domain. By leveraging the FieldSplit structure, developed in Section 2, we were able to precondition the interior regions and interface regions separately, permitting us to exploit the low-rank nature of the differentiation matrices used to enforce the interface conditions. This allows meshfree coupling to maintain computational viability despite the otherwise dense matrix produced during Gaussian interpolation.

Looking forward, several issues still need to be addressed. The required minimum length of the eigenfunction series to optimally recover the function is unknown, and using a larger M incurs cost which is not providing any additional performance. This is an open problem for any scattered data approximation problem using GaussQRr. In Chapter 4 numerical results were presented that for some functions, regardless of the number of input points N, there is only a fixed M necessary to recover the function to some desired accuracy. One possible technique

for these coupled simulations would be to experiment on small grids, with small computational cost, and try to find that M. Once the M is known on that small grid, it may still be appropriate even for much larger systems.

Along with the issue of the free parameter M is the free parameter ε which must be chosen appropriately for the MF coupling scheme to outperform the FD coupling scheme. This has frustrated the approximation theory and statistics communities for some time and is not any simpler in the multiphysics coupling application. Figure 6.5b seems to indicate that knowledge of a good ε on small grids can be used to find a good ε on large grids, although that may only be true for this specific simulation. The most well-founded approach to determining ε probably involves statistics (e.g., cross-validation, maximum likelihood estimation, inference), and is addressed for the scattered data approximation problem in Chapter 8.

In Section 6.3.2 we discussed the choice of eigenfunctions included in the simulation; specifically, the degree of the eigenfunctions in the x direction was bounded by the width of the coupling region. Because there is a natural preference between dimensions, it is also natural to consider anisotropic kernels – Gaussians with different shape parameters ε_x and ε_y in the x and y dimensions respectively. Such a situation can be handled naturally given the tensor product structure of the Gaussian, as described in Chapter 4. This approach has the unpleasant effect of adding another free parameter to the kernel, thus exacerbating the problems associated with determining ε in the isotropic case. The benefit would be increased potential accuracy, because the anisotropic parameters could always be fixed $\varepsilon_x = \varepsilon_y$ yielding the isotropic case as the upper bound on possible error.

Even though we have addressed the computational cost of performing matrix vector products involving the differentiation matrices $\mathbf{\Phi}^{\mathcal{D}_k}\mathbf{\Phi}^{+}$, we have not yet

addressed the cost of computing Φ^+. The dominant cost of many least squares solvers is the QR factorization, which here costs $\mathcal{O}(NM^2)$. As already mentioned, we are interested in keeping M small, which help alleviates some of the cost of this solve. Another possible approach to improving the speed of the least squares solve is the $\mathcal{O}(NM)$ QR factorization method derived in Chapter 5, though that method only for 1D GaussQRr problems, so far.

CHAPTER 7

SOLVING BOUNDARY VALUE PROBLEMS WITH EIGENFUNCTIONS

7.1 Introduction

In Chapter 4, we developed a stable method for solving the scattered data approximation problem in $\mathbb{R}^d$ using increasingly flat Gaussians, for which traditional methods were susceptible to severe ill-conditioning. This concept was extended to approximating derivatives of scattered data in Chapter 6, and we demonstrated how multiphysics simulations could be coupled together stably and accurately using this approach. One of the benefits of this meshfree coupling was that the individual discretizations of each component were immaterial to the interpolation strategy.

Another way to solve this problem would be to instead discretize the component simulations in such a way that the relevant information could be easily extracted/computed on the coupling interface. One possible method for doing this would be to use our stable Gaussian expansion as a basis for the boundary value problem (BVP)solution. This chapter introduces and develops that idea, presenting GaussQR as a standalone method for solving BVP, as well as an approximation component within a larger BVP solver. Adding this tool to our multiphysics infrastructure provides the ability to spatially discretize simulations with a method that simplifies the multiphysics coupling.

Section 7.2 discusses the current state of kernel methods for solving BVPs. In Section 7.3 we consider the solution of boundary value problems by collocation

with traditional Gaussian kernels, and demonstrate the benefit of instead using the eigenfunction expansion. We also consider the use of differentiation matrices [168] to solve problems. In Section 7.4 the eigenfunction expansion is applied to approximate particular solutions and solve BVPs with the method of particular solutions. We extend this particular solution approach in Section 7.4.3 to incorporate boundary data and produce a more accurate solution at less cost. This chapter is based significantly on

M. McCourt, *Stable Gaussians for boundary value problems*, Advanced in Applied Mathematics and Mechanics, accepted

which is cited as [123].

7.2 Existing kernel methods for boundary value problems

Kernel-based meshfree approximation methods have gained popularity in several fields, including scattered data interpolation [176], finance [93], statistics [163], machine learning [137] and others. One of the great benefits of using these methods is that no discretization of the relevant domain is required; basis functions are centered at various points throughout the domain, allowing for kernel-based methods to circumvent some of the barriers associated with higher dimensional problems. Additionally, a variety of kernels exist, providing users in each application the ability to tailor the solution basis to fit that application's specific opportunities and constraints.

Techniques for solving BVPs with radial basis functions (RBFs) have advanced significantly in the past two decades. The original method for solving elliptic partial differential equations (PDEs) with RBFs came in 1990 [103] and involved an unsymmetric collocation of basis functions at points chosen throughout the domain. Since that initial work, further analysis has been done on the convergence of this collocation method [152], which has encouraged its use despite its theoretic potential for failure [94]. A symmetric collocation technique was also developed [59] which ensured invertibility of the collocation system by using a modified set of basis functions.

Another popular method for solving BVP with radial basis functions is the method of fundamental solutions [55]. Essentially, this method replaces the BVP with an interpolation problem on the boundary using functions which satisfy the PDE. The mathematical formulation of this method is well-developed, but it is only applicable for homogeneous problems where the fundamental solution is known. The method of particular solutions [40] is an adaptation for inhomogeneous problems involving two approximation systems: one to satisfy the inhomogeneity in the interior, and another to satisfy the boundary conditions, assuming a now homogeneous problem. The use of radial basis functions to approximate particular solutions was discussed in [82, 96].

One of the great shortcomings of radial basis functions is that, for some parameterizations, the resulting linear system may be irrevocably ill-conditioned [60]. Even more troublesome is the fact that the most accurate parameterizations may lie in the ill-conditioned regime [74]. This ill-conditioning is especially significant for kernels with a great deal of smoothness, which often tempers the optimism of researchers hoping to exploit their spectral accuracy. In Chapter 4, this obstacle

was addressed for scattered data interpolation problems using Gaussians in $\mathbb{R}^d$ by exploiting a truncated eigenfunction expansion of the Gaussian. Here, we extend the approximation via eigenfunctions to the solution of boundary value problems.

There are many methods for solving boundary value problems with kernels that are not discussed in this thesis. Multilevel methods [117, 100] have been presented, including for higher order problems [2], to attempt to mitigate the cost associated with solving dense systems generated by globally supported RBFs. Finite difference schemes based on RBFs [67, 68] have proven to be an effective meshfree solver for geological and climate based problems. Partition of unity methods [114] are being developed now to incorporate RBF collocation with other solution schemes for applications including crack propagation. Petrov-Galerkin techniques [6] have been developed to allow the weak form solution of PDEs, while recent work [154] has provided analytic support for this approach. Some work has been done incorporating RBFs into discontinuous Galerkin schemes [141]. Kernel based PDE solvers on manifolds [77] are beginning to mature as well. To narrow our focus from all possible BVP solvers using kernels, we only discuss collocation and the method of particular solutions.

7.3 Collocation using Gaussian eigenfunctions

The original RBF collocation technique in [103] involved multiquadrics supplemented by linear polynomials. These basis functions are subject to severe ill-conditioning depending on the flatness of the multiquadrics. This ill-conditioning is the result of extremely flat basis functions looking too much alike, causing the representative columns in the collocation matrix to become indistinguishable and

making the system appear to be low rank.

This problem is not unique to multiquadrics or to collocation techniques; indeed any application requiring the inversion of matrices generated by very smooth RBFs will fall victim to this as the RBFs approach their flat limit. It has been discussed for interpolation problems that, despite this perceived impasse, the problem itself is not necessarily ill-conditioned [115, 52, 134]. Rather, it is the solution approach (i.e., forming a linear system using the RBF basis) which deals the damage [113], and if an alternate method could handle the ill-conditioning the true solution could be found [74].

One such approach to solving this problem is to find a series expansion for the kernel which allows for the removal of the ill-conditioned terms analytically. This solution technique is called RBF-QR [72], and it has been used successfully on the circle/sphere for both interpolation [70] and PDEs [73]. In these papers, the authors discussed the possibility that the most accurate kernel parameterizations were also too ill-conditioned to treat directly, necessitating the series expansion approach.

In Chapter 4, a series expansion was developed to allow for stable approximation with Gaussians in $\mathbb{R}^d$; this expansion was based on the eigenfunctions of the associated Hilbert-Schmidt operator. Because the Gaussian kernel in higher dimensions is formed through tensor products, the higher dimensional series expansion is also formed with a tensor product, trivially allowing the move to $\mathbb{R}^d$. The approximation of derivatives using this series expansion was discussed in the previous chapter. Here we would like to use these derivatives to solve boundary value problems with collocation.

7.3.1 Ill-conditioning in Gaussian basis collocation

Linear BVPs, without dependence on time, can generally be phrased in the form

$$\begin{aligned} \mathcal{L}u &= f, \qquad \text{on the interior } \Omega, \\ \mathcal{B}u &= g, \qquad \text{on the boundary } \partial\Omega, \end{aligned}$$

where $\mathcal{L}$ is the linear PDE operator, and $\mathcal{B}$ is the linear boundary condition operator. $\Omega \in \mathbb{R}^d$ is a bounded domain with Lipschitz boundary. In unsymmetric kernel collocation, we assume that the solution takes the form

$$u(\boldsymbol{x}) = \sum_{k=1}^{N} a_k K(\boldsymbol{x}, \boldsymbol{x}_k) + \sum_{\ell=1}^{q} a_{N+\ell} p_\ell(\boldsymbol{x}) \tag{7.1}$$

where $\boldsymbol{x} \in \partial\Omega \cup \Omega$ is a d-dimensional vector for a problem in $\mathbb{R}^d$, $\{\boldsymbol{x}_k\}_{k=1}^N$ are the kernel centers, N is the number of kernels used, K is the kernel, $\{p_\ell\}_{\ell=1}^q$ are polynomial terms, and q is the number of polynomial terms. For this thesis, we assume that no polynomial terms are necessary, though later we briefly discuss the effect this may have on the accuracy of the solution and optimal choice of K.

Choosing $q = 0$ removes the polynomial terms and leave the pure kernel series

$$u(\boldsymbol{x}) = \sum_{k=1}^{N} a_k K(\boldsymbol{x}, \boldsymbol{x}_k). \tag{7.2}$$

This matches the form of the solution to the scattered data interpolation problem as defined in (1.8). Assuming that we have chosen $N_{\mathcal{L}}$ collocation points on the interior and $N_{\mathcal{B}}$ collocation points on the boundary, we can now apply the BVP operators to (7.2); note that the PDE operators act on the first kernel argument, as the second kernel argument defines the center of the kernel, not where the kernel

is being evaluated. This leaves us with the continuous collocation equations

$$\sum_{k=1}^{N} a_k \mathcal{L}K(\boldsymbol{x}, \boldsymbol{x}_k) = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega,$$
$$\sum_{k=1}^{N} a_k \mathcal{B}K(\boldsymbol{x}, \boldsymbol{x}_k) = g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \partial\Omega.$$

We must now choose a finite number of points, $N_{\mathcal{L}}$ on the interior and $N_{\mathcal{B}}$ on the boundary, at which to enforce these equations. If the $\{\boldsymbol{x}_k\}_{k=1}^{N_{\mathcal{L}}}$ interior points are ordered before the $\{\boldsymbol{x}_k\}_{k=1+N_{\mathcal{L}}}^{N_{\mathcal{B}}+N_{\mathcal{L}}}$ boundary points, this system of linear equations has the matrix form

$$\begin{pmatrix} \mathcal{L}K(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \mathcal{L}K(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ & \vdots & \\ \mathcal{L}K(\boldsymbol{x}_{N_{\mathcal{L}}}, \boldsymbol{x}_1) & \cdots & \mathcal{L}K(\boldsymbol{x}_{N_{\mathcal{L}}}, \boldsymbol{x}_N) \\ \mathcal{B}K(\boldsymbol{x}_{N_{\mathcal{L}}+1}, \boldsymbol{x}_1) & \cdots & \mathcal{B}K(\boldsymbol{x}_{N_{\mathcal{L}}+1}, \boldsymbol{x}_N) \\ & \vdots & \\ \mathcal{B}K(\boldsymbol{x}_{N_{\mathcal{L}}+N_{\mathcal{B}}}, \boldsymbol{x}_1) & \cdots & \mathcal{B}K(\boldsymbol{x}_{N_{\mathcal{L}}+N_{\mathcal{B}}}, \boldsymbol{x}_N) \end{pmatrix} \begin{pmatrix} a_1 \\ \\ \vdots \\ \vdots \\ \\ a_N \end{pmatrix} = \begin{pmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_{N_{\mathcal{L}}}) \\ g(\boldsymbol{x}_{N_{\mathcal{L}}+1}) \\ \vdots \\ g(\boldsymbol{x}_{N_{\mathcal{L}}+N_{\mathcal{B}}}) \end{pmatrix} \tag{7.3}$$

By choosing $N_{\mathcal{L}} + N_{\mathcal{B}} = N$, the system (7.3) is square, and if it is nonsingular [152] it has a unique solution.

Theoretically, there is nothing requiring the kernel centers to be the same as the collocation points. We consider no such instances here, although such material is presented for interpolation in [69] and PDEs [66, 159] suggesting that this may improve the error near the boundary. By choosing the kernel centers to match the collocation points, we trivially satisfy $N_{\mathcal{L}} + N_{\mathcal{B}} = N$ and must solve a square linear system to find $a_1, \ldots, a_N$.

To demonstrate their notoriously ill-conditioned behavior, and in following with the results presented in Chapter 4, we consider Gaussian kernels

$$K(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\varepsilon^2 \|\boldsymbol{x} - \boldsymbol{z}\|^2)$$

for the collocation solution. The value ε is the *shape parameter*, so-called because for large ε the Gaussians become very peaked, and for small ε the Gaussians become very flat. A well-chosen ε can allow for very accurate solutions (even more accurate than polynomials in some cases) whereas a poorly chosen ε may provide little or no accuracy. See Figure 7.1 for a demonstration of the effect ε can have on accuracy.

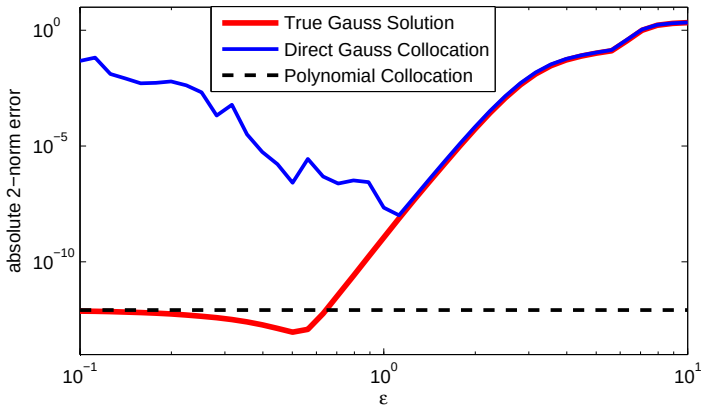


Figure 7.1: Solving (7.3) produces a good solution until ill-conditioning overwhelms the accuracy, preventing the solution from reaching its polynomial limit. If we could stably solve the system, we should find the "True Gauss Solution" curve. Error is computed at 200 evenly spaced points in the domain.

Figure 7.1 was generated by solving the boundary value problem

$$u_{xx}(x) = \frac{-\sinh(x)}{(1+\cosh(x))^2}, \quad x \in (-1,1), \tag{7.4a}$$

$$u(x) = \frac{\sinh(x)}{1+\cosh(x)}, \qquad x \in \{-1,1\}, \tag{7.4b}$$

with $N = 16$ collocation points located at the Chebyshev nodes

$$x_k = \cos\left(\pi\frac{(k-1)}{N-1}\right), \qquad 1 \le k \le N.$$

Phrased in terms of the general BVP language from earlier, this problem has

components

$$\mathcal{L} = \frac{\mathrm{d}^2}{\mathrm{d}x^2}, \qquad f(x) = \frac{-\sinh(x)}{(1+\cosh(x))^2},$$
$$\mathcal{B} = \mathcal{I}, \qquad g(x) = \frac{\sinh(x)}{1+\cosh(x)},$$

where $\mathcal{I}$ is the identity operator, $\mathcal{I}u = u$. The solution named "Direct Gauss Collocation" was computed by solving the system (7.3) using the standard Gaussian basis. The poor behavior as $\varepsilon \to 0$ is a result of the ill-conditioning in the collocation matrix: for $\varepsilon = 1$ the condition number is $\mathcal{O}(10^{13})$, even though the matrix is only size $N = 16$.

It has been proven [70] that this ill-conditioning is a symptom only of the choice of basis, and not fundamental to the approximation problem. For interpolation problems we have seen that the limit of Gaussian as $\varepsilon \to 0$ is well-defined, and is in fact equal to the polynomial interpolant Chapter 4. We therefore expect that, in the absence of ill-conditioning, the Gaussian collocation solution would approach the "Polynomial Collocation" result; this polynomial solution was computed using the differentiation matrix approach from [168]. The "True Gauss Solution" displayed above shows this desired behavior; we now explain how this solution is computed without the ill-conditioning inherent in solving (7.3).

7.3.2 Collocation using the stable basis

We need to replace the kernel $K(x, z) = \mathrm{e}^{-\varepsilon^2|x-z|^2}$ with its truncated eigenfunction expansion

$$\mathrm{e}^{-\varepsilon^2|x-z|^2} = \sum_{k=1}^{M} \lambda_k \varphi_k(x) \varphi_k(z),$$

where λ_k and φ_k are

$$\lambda_k = \sqrt{\frac{\alpha^2}{\alpha^2 + \delta^2 + \varepsilon^2}} \left(\frac{\varepsilon^2}{\alpha^2 + \delta^2 + \varepsilon^2} \right)^{k-1}, \tag{4.4b}$$

$$\varphi_k(x) = \gamma_k \mathrm{e}^{-\delta^2 x^2} H_{k-1}(\beta \alpha x), \tag{4.4a}$$

with H_{k-1} the degree $k-1$ Hermite polynomial. The value α is the global scale parameter as defined in Section 4.3, and the auxiliary parameters

$$\beta = \left(1 + \left(\frac{2\varepsilon}{\alpha} \right)^2 \right)^{\frac{1}{4}}, \quad \gamma_k = \sqrt{\frac{\beta}{2^{k-1}\Gamma(k)}}, \quad \delta^2 = \frac{\alpha^2}{2} \left(\beta^2 - 1 \right),$$

are defined in terms of ε and α. The truncation value is assumed to satisfy $M > N$, although this assumption is reconsidered later. The value M is chosen large enough to satisfy a bound on the ratio λ_M / λ_N; this choice is described in Section 4.4.2, and is be discussed here. Regardless of the value of M, the eigenfunction series is the optimal M-term approximation to the Gaussian in the $L_2(\mathbb{R}, \rho)$ sense, where

$$\rho(x) = \frac{\alpha}{\sqrt{\pi}} \, \mathrm{e}^{-\alpha^2 x^2}$$

is a weight function which localizes the L_2 inner product [137].

In matrix form, this M-term series expansion can be written as

$$\mathrm{e}^{-\varepsilon^2 |x-z|^2} = \begin{pmatrix} \varphi_1(x) & \dots & \varphi_M(x) \end{pmatrix} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_M \end{pmatrix} \begin{pmatrix} \varphi_1(z) \\ \vdots \\ \varphi_M(z) \end{pmatrix}.$$

Substituting this into the matrix from (7.3), and noting that the operators $\mathcal{L}$ and

$\mathcal{B}$ apply to the first kernel argument, converts that matrix to

$$\begin{pmatrix} \mathcal{L}\varphi_1(x_1) & \cdots & \mathcal{L}\varphi_M(x_1) \\ & \vdots & \\ \mathcal{L}\varphi_1(x_{N_\mathcal{L}}) & \cdots & \mathcal{L}\varphi_M(x_{N_\mathcal{L}}) \\ \mathcal{B}\varphi_1(x_{N_\mathcal{L}+1}) & \cdots & \mathcal{B}\varphi_M(x_{N_\mathcal{L}+1}) \\ & \vdots & \\ \mathcal{B}\varphi_1(x_{N_\mathcal{L}+N_\mathcal{B}}) & \cdots & \mathcal{B}\varphi_M(x_{N_\mathcal{L}+N_\mathcal{B}}) \end{pmatrix} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_M \end{pmatrix} \begin{pmatrix} \varphi_1(x_1) & \cdots & \varphi_1(x_N) \\ & \vdots & \\ \varphi_M(x_1) & \cdots & \varphi_M(x_N) \end{pmatrix}. \tag{7.5}$$

This allows (7.3) to be written in block form as

$$\begin{pmatrix} \mathcal{L}\Phi_{\mathcal{L},1} & \mathcal{L}\Phi_{\mathcal{L},2} \\ \mathcal{B}\Phi_{\mathcal{B},1} & \mathcal{B}\Phi_{\mathcal{B},2} \end{pmatrix} \begin{pmatrix} \Lambda_1 & \\ & \Lambda_2 \end{pmatrix} \begin{pmatrix} \Phi_{\mathcal{L},1}^T & \Phi_{\mathcal{B},1}^T \\ \Phi_{\mathcal{L},2}^T & \Phi_{\mathcal{B},2}^T \end{pmatrix} \boldsymbol{a} = \begin{pmatrix} \boldsymbol{f}_\mathcal{L} \\ \boldsymbol{g}_\mathcal{B} \end{pmatrix}, \tag{7.6}$$

where

$$\begin{aligned}
(\Phi_{\mathcal{L},1})_{j,k} &= \varphi_k(x_j) && \text{for} && 1 \leq k \leq N,\ x_j \in \Omega, \\
(\Phi_{\mathcal{L},2})_{j,k} &= \varphi_k(x_j) && \text{for} && N+1 \leq k \leq M,\ x_j \in \Omega, \\
(\Phi_{\mathcal{B},1})_{j,k} &= \varphi_k(x_j) && \text{for} && 1 \leq k \leq N,\ x_j \in \partial\Omega, \\
(\Phi_{\mathcal{B},2})_{j,k} &= \varphi_k(x_j) && \text{for} && N+1 \leq k \leq M,\ x_j \in \partial\Omega, \\
(\Lambda_1)_{k,k} &= \lambda_k && \text{for} && 1 \leq k \leq N, \\
(\Lambda_2)_{k,k} &= \lambda_{k+N} && \text{for} && 1 \leq k \leq M-N, \\
(\boldsymbol{f}_\mathcal{L})_j &= f(x_j) && \text{for} && x_j \in \Omega, \\
(\boldsymbol{g}_\mathcal{B})_j &= g(x_j) && \text{for} && x_j \in \partial\Omega.
\end{aligned}$$

For terms such as $\mathcal{L}\Phi_{\mathcal{L},1}$ which appear in (7.6), the operator passes through naturally using the matrix definitions above: $(\mathcal{L}\Phi_{\mathcal{L},1})_{j,k} = \mathcal{L}\varphi_k(x_j)$.

As discussed in [72], the ill-conditioning in this system exists primarily in the diagonal matrix containing Λ_1 and Λ_2. The RBF-QR approach to alleviating this

ill-conditioning is described in Section 4.4 and converts the symmetric positive definite system (7.6) to the unsymmetric (but still nonsingular) system

$$\begin{pmatrix} \mathcal{L}\Phi_{\mathcal{L},1} & \mathcal{L}\Phi_{\mathcal{L},2} \\ \mathcal{B}\Phi_{\mathcal{B},1} & \mathcal{B}\Phi_{\mathcal{B},2} \end{pmatrix} \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2(\Phi_{\mathcal{L},2}^T \;\; \Phi_{\mathcal{B},2}^T)(\Phi_{\mathcal{L},1}^T \;\; \Phi_{\mathcal{B},1}^T)^{-1}\Lambda_1^{-1} \end{pmatrix} \hat{\boldsymbol{a}} = \begin{pmatrix} \boldsymbol{f}_{\mathcal{L}} \\ \boldsymbol{g}_{\mathcal{B}} \end{pmatrix}. \tag{7.7}$$

The Λ_2 and Λ_1^{-1} terms can be applied simultaneously, preventing overflow or underflow issues. Because the Λ_2 terms are exponentially smaller than the Λ_1 terms (refer to (4.4b)) there are no fears about this new formulation undergoing dangerous growth. The term $(\Phi_{\mathcal{L},2}^T \;\; \Phi_{\mathcal{B},2}^T)(\Phi_{\mathcal{L},1}^T \;\; \Phi_{\mathcal{B},1}^T)^{-1}$ is generally computed using the QR factorization (thus the name RBF-QR) to avoid mixing different orders of the eigenfunctions during the decomposition. We refer to this eigenfunction approach, joint with RBF-QR, as **GaussQR**, as we did for the interpolation problem in Chapter 6.

The new coefficients $\hat{\boldsymbol{a}}$ can be related to the standard Gaussian basis coefficients $\boldsymbol{a}$ by

$$\Lambda_1(\Phi_{\mathcal{L},1}^T \;\; \Phi_{\mathcal{B},1}^T)\boldsymbol{a} = \hat{\boldsymbol{a}},$$

but computing $\boldsymbol{a}$ is not recommended; the Λ_1 matrix is severely ill-conditioned because of the exponentially decreasing eigenvalues. Because of this, we solve for and evaluate the interpolant only in terms of the stable basis $\{\psi_k\}_{k=1}^N$:

$$\begin{aligned} u(x) &= \boldsymbol{\psi}(x)^T\hat{\boldsymbol{a}} \\ &= (\psi_1(x) \quad \cdots \quad \psi_N(x))\hat{\boldsymbol{a}} \\ &= (\varphi_1(x) \quad \cdots \quad \varphi_M(x)) \begin{pmatrix} \mathsf{I}_N \\ \Lambda_2(\Phi_{\mathcal{L},2}^T \;\; \Phi_{\mathcal{B},2}^T)(\Phi_{\mathcal{L},1}^T \;\; \Phi_{\mathcal{B},1}^T)^{-1}\Lambda_1^{-1} \end{pmatrix} \hat{\boldsymbol{a}}. \end{aligned} \tag{7.8}$$

By applying the specific BVP operators and functions described above, solving the system (7.7), and evaluating the solution with (7.8), we can generate the "True

Gauss Solution" curve presented in Figure 7.1. That solution matches the standard basis solution for larger values of ε, and achieves the expected polynomial limit as $\varepsilon \to 0$. The global scale parameter α was set to 1 for these experiments.

7.3.3 Low-rank series approximate collocation

In order to produce the stable collocation solution in Section 7.3.2, the eigenfunction series must be chosen with $M > N$. As discussed in Chapter 4, it may be possible to choose $M < N$ when N is large or for $\varepsilon \ll 1$. This is especially important in higher dimensions, where satisfying $\lambda_M/\lambda_N < \epsilon_{\text{mach}}$ for $\epsilon_{\text{mach}} \approx 10^{-16}$ requires more eigenfunctions depending on the dimension of the problem.

This shift to an early truncation point $M < N$ has a significant change on the collocation problem, because it converts the full-rank system (7.6) into a rank M system. Properties of this low rank system were discussed in Section 4.6 and Chapter 5, although only for the scattered data approximation problem. The transition follows the same pattern as before, except using a low-rank approximation to the Gaussian. Starting from (7.5), and imposing the restriction $M < N$ produces the rank-M collocation system

$$\begin{pmatrix} \mathcal{L}\Phi_{\mathcal{L}} \\ \mathcal{B}\Phi_{\mathcal{B}} \end{pmatrix} \Lambda \begin{pmatrix} \Phi_{\mathcal{L}}^T & \Phi_{\mathcal{B}}^T \end{pmatrix} \boldsymbol{a} = \begin{pmatrix} \boldsymbol{f}_{\mathcal{L}} \\ \boldsymbol{g}_{\mathcal{B}} \end{pmatrix}.$$

We use similar block definitions as before, with

$$\begin{aligned}
(\Phi_{\mathcal{L}})_{j,k} &= \varphi_k(x_j) \qquad \text{for} \qquad 1 \leq k \leq M,\ x_j \in \Omega, \\
(\Phi_{\mathcal{B}})_{j,k} &= \varphi_k(x_j) \qquad \text{for} \qquad 1 \leq k \leq M,\ x_j \in \partial\Omega, \\
(\Lambda)_{k,k} &= \lambda_k \qquad \text{for} \qquad 1 \leq k \leq M, \\
(\boldsymbol{f}_{\mathcal{L}})_j &= f(x_j) \qquad \text{for} \qquad x_j \in \Omega, \\
(\boldsymbol{g}_{\mathcal{B}})_j &= g(x_j) \qquad \text{for} \qquad x_j \in \partial\Omega.
\end{aligned}$$

This system is still as ill-conditioned as the Λ matrix, so we redefine the system as

$$\begin{pmatrix} \mathcal{L}\Phi_{\mathcal{L}} \\ \mathcal{B}\Phi_{\mathcal{B}} \end{pmatrix} \tilde{\boldsymbol{a}} = \begin{pmatrix} \boldsymbol{f}_{\mathcal{L}} \\ \boldsymbol{g}_{\mathcal{B}} \end{pmatrix}, \tag{7.9}$$

with

$$\Lambda \begin{pmatrix} \Phi_{\mathcal{L}}^T & \Phi_{\mathcal{B}}^T \end{pmatrix} \boldsymbol{a} = \tilde{\boldsymbol{a}}.$$

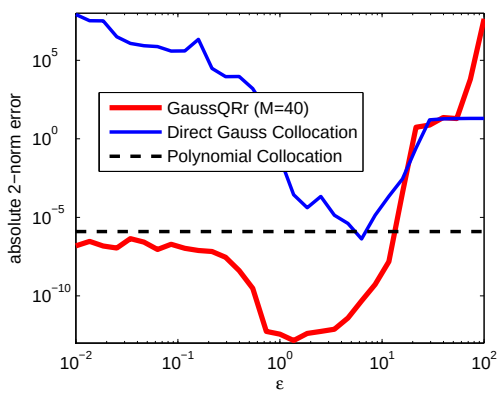
This allows us to avoid inverting Λ, as long as we work in the new basis $\{\varphi_k\}_{k=1}^M$, which is just the first M eigenfunctions.

Because (7.9) is a system of N equations in $M < N$ unknowns, there is likely no consistent solution. Instead, $\tilde{\boldsymbol{a}}$ must be determined in a least squares sense. We have named this low-rank solution method **GaussQRr** (recall Section 6.1) because a regression system is solved instead of a square system. This method is tested on the BVP

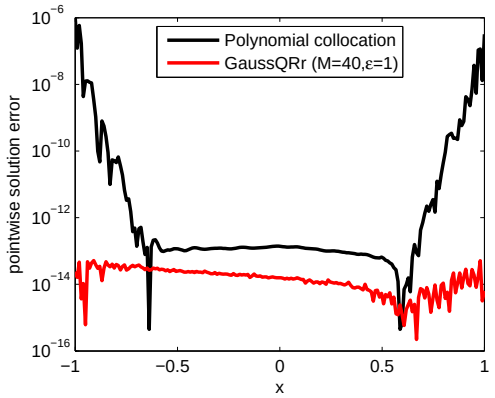
$$u_{xx}(x) = -9\pi^2 \sin(3\pi x) - \pi^2 \cos(\pi x), \quad x \in (-1, 1), \tag{7.10a}$$

$$u(x) = \sin(3\pi x) + \cos(\pi x) + 1, \qquad x \in \{-1, 1\}, \tag{7.10b}$$

using $N = 80$ collocation points at the Chebyshev nodes. See Figure 7.2 to compare this method to the other methods "Polynomial Collocation" and "Direct Gauss Collocation" which we have previously used.



(a) Using GaussQRr, for some ε values, we can achieve many orders of magnitude more accuracy than with polynomial collocation (of degree N).



(b) As is often the case, the polynomial solution error is concentrated at the boundaries. This contrasts with the evenly spread error for GaussQRr.

Figure 7.2: The GaussQRr method is an effective approach to solving the BVP (7.10a) for small ε. Parameter values $\alpha = 1$ and $M = .5N = 40$ were used for these experiments. Error is computed at 200 evenly spaced points in the domain.

In Figure 7.2a, we can see again that the Gaussian collocation solution computed in the Gaussian basis becomes ill-conditioned very quickly, preventing it from reaching its optimal accuracy. The GaussQRr method, performed here with $M = 40$, can find solutions with many orders of magnitude more accuracy than any directly computed solution. The "Polynomial Collocation" solution is displayed only for reference; because $M < N$, we no longer expect the limit of the GaussQRr solution to match the degree N polynomial result. Additionally, we cannot trust solutions of GaussQRr for large values of ε because the eigenvalues (4.4a) decay less quickly and our truncation assumption becomes less valid.

One of the positive outcomes of the GaussQRr solution is that the error is more evenly distributed throughout the domain. Figure 7.2 shows that the $\varepsilon = 1$ GaussQRr pointwise error at all $x \in [-1, 1]$ is roughly $\mathcal{O}(10^{-14})$, in contrast to the "Polynomial collocation" pointwise error which is significantly greater near the boundaries. The effect of point distribution is not discussed here as that is a much

too complicated topic; studies on this include [69] for interpolation and [119, 149] for PDEs. We note only that the points chosen here tend to be clustered near the boundary, as suggested in [168] for the polynomial collocation technique.

7.3.4 A nonlinear time stepping example

Thus far we have presented only linear examples, but the Gaussian eigenfunction expansion can also be exploited for nonlinear problems. When choosing $M > N$, this yields a nonlinear system of N equations, and when $M < N$, this yields a nonlinear least squares problem in M unknowns. We consider an example using GaussQRr in this section.

For this section we choose to solve the linear critical gradient equation introduced in Section 3.4. As a reminder, it can be written in 1D as

$$u_t - (\kappa(u_x)u_x)_x = f, \qquad x \in (-1,1),\ t > 0 \tag{7.11a}$$

$$u = g, \qquad x \in \{-1,1\} \tag{7.11b}$$

$$u = u_0, \qquad t = 0 \tag{7.11c}$$

where the diffusivity κ is a function of the derivative u_x

$$\kappa(u_x) = \frac{\mu}{2\tau}\log(\cosh(2\tau u_x) + \cosh(2\tau C)) - \mu C + \frac{\mu - 2}{2\tau}\log(2) + \kappa_0 - B.$$

All experiments here use the parameter values

$$\mu = 10, \qquad \tau = 1, \qquad C = .5, \qquad \kappa_0 = 1.$$

and B is just an integration constant so that $\kappa(0) = \kappa_0$.

In a plasma physics setting, the source term $f(x) = \mathrm{e}^{-x}$ would be used to cause a pedestal to form at the magnetic separatrix. While this problem is useful for

modeling magnetic confinement fusion, it is less useful for studying the accuracy of the numerical scheme because there is no analytic solution for that source. Instead, we choose a solution which has a pedestal-like shape,

$$u(x,t) = \operatorname{erf}(4(1-\mathrm{e}^{-t})x) + 1,$$

which also defines the functions

$$\begin{aligned} g(x,t) &= \operatorname{erf}(4(1-\mathrm{e}^{-t})x) + 1, \\ u_0(x) &= 1. \end{aligned}$$

For the GaussQRr approximation, we require our solution to take the form

$$\hat{u}(x,t) = \sum_{k=1}^{M} a_k(t)\varphi_k(x) = (\varphi_1(x) \ \cdots \ \varphi_M(x)) \begin{pmatrix} a_1(t) \\ \vdots \\ a_M(t) \end{pmatrix} = \boldsymbol{\phi}(x)^T \boldsymbol{a}(t).$$

We choose $N-2$ collocation points on the interior, and require $x_{N-1} = -1$ and $x_N = 1$ to satisfy the boundary conditions. This can now be substituted back into (7.11a) to yield the system of nonlinear ODEs

$$\begin{aligned} \boldsymbol{\phi}(x_j)^T \boldsymbol{a}_t(t) - \boldsymbol{\phi}_{xx}(x_j)^T \boldsymbol{a}(t) \Big[\kappa' \left(\boldsymbol{\phi}_x(x_j)^T \boldsymbol{a}(t) \right) \boldsymbol{\phi}_x(x_j)^T \boldsymbol{a}(t) + \\ \kappa \left(\boldsymbol{\phi}_x(x_j)^T \boldsymbol{a}(t) \right) \Big] = f(x_j,t), \end{aligned}$$

for $1 \leq j \leq N-2$. At this point, we are no longer writing the problem in its conservative form; this is hardly a problem though, since by using Gaussians we have already assumed that the solution is in the Gaussian native space, and thus has enough smoothness to justify the second derivative. Adding in the 2 equations from the boundary conditions (7.11c),

$$\boldsymbol{\phi}(x_j)^T \boldsymbol{a}(t) - g(x_j,t) = 0,$$

for $j = N-1, N$ gives a system of N differential algebraic equations.

We choose here to discretize in time using the backwards Euler formula, although this choice is made more for simplicity than for any computational benefit. This leaves us with the nonlinear system of equations

$$\begin{aligned} \boldsymbol{\phi}(x_j)^T \left[\frac{\boldsymbol{a}^n - \boldsymbol{a}^{n-1}}{\Delta t}\right] - \boldsymbol{\phi}_{xx}(x_j)^T \boldsymbol{a}^n \Big[\kappa'\big(\boldsymbol{\phi}_x(x_j)^T \boldsymbol{a}^n\big)\, \boldsymbol{\phi}_x(x_j)^T \boldsymbol{a}^n + \\ \kappa\big(\boldsymbol{\phi}_x(x_j)^T \boldsymbol{a}^n\big)\Big] - f(x_j, t_n) = 0 \qquad (7.12) \\ \boldsymbol{\phi}(x_j)^T \boldsymbol{a}^n - g(x_j, t) = 0 \end{aligned}$$

where, at each time step t_n, the solution is $\boldsymbol{a}^n$. The initial condition $\boldsymbol{a}^0$ is computed by solving the GaussQRr approximation problem

$$\begin{pmatrix} \boldsymbol{\phi}(x_1)^T \\ \vdots \\ \boldsymbol{\phi}(x_N)^T \end{pmatrix} \boldsymbol{a}^0 = \begin{pmatrix} u_0(x_1, 0) \\ \vdots \\ u_0(x_N, 0) \end{pmatrix}.$$

At each time step t_k we need to solve a nonlinear least squares problem with N equations and M unknowns, the $\boldsymbol{a}(t_k)$. For the initial guess at each time step, we solve the system (7.12) with $\kappa \equiv 1$, which reduces the problem to a linear least squares system. Error results are displayed in Figure 7.3.

These experiments confirm that, at least for this example, the separation of spatial and temporal discretizations is appropriate. This so-called method of lines approach has not affected the accuracy of the backward Euler method, which converges with its standard order $\mathcal{O}(\Delta t)$. The convergence terminates when the error introduced by the spatial discretization dominates, which occurs for increasingly accurate solutions as N is increased. Moreover, the GaussQRr solver appears to maintain its spectral convergence, subject to the accuracy bound imposed by the time stepping. Obviously, we have only tested it here for relatively small N, so

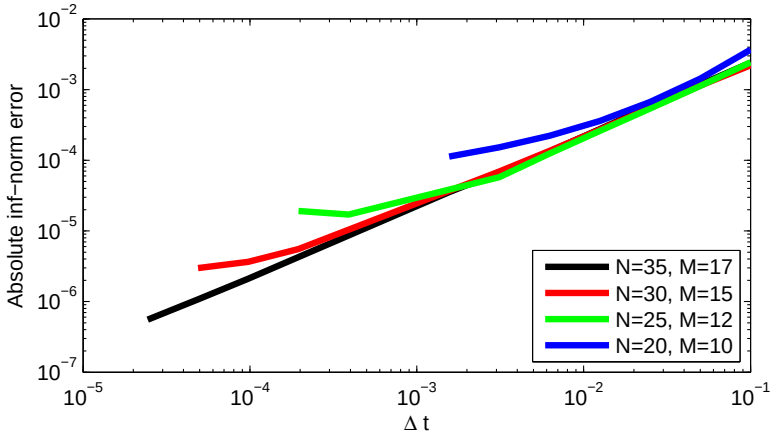


Figure 7.3: The error in the time stepping is bounded either by the $\mathcal{O}(\Delta t)$ error of the Euler discretization, or the GaussQRr accuracy. When the solution levels off the collocation error has become the dominant term. For all experiments, GaussQRr used the parameters $M = .5N$, $\varepsilon = 10^{-2}$, $\alpha = 1$. Collocation points are evenly spaced in the domain, and the error is computed at the collocation points at $t = .5$.

further study will be needed for more complicated time dependent problems. It will also be useful to consider problems involving $M > N$, where the GaussQR collocation technique results in square nonlinear systems at each time step.

7.3.5 Solving problems with a differentiation matrix

The examples up until now have only solved problems in one spatial dimension, but with only minor notational corrections these techniques are valid in arbitrary dimensions. Various technical considerations for moving to higher dimensions are discussed in Chapter 4. In this chapter, the only significant change is the change in the definition of eigenfunctions from their 1D form to their tensor product form. This means that the kernel in $\mathbb{R}^d$ would now take the form

$$\mathrm{e}^{-\varepsilon^2\|\boldsymbol{x}-\boldsymbol{z}\|^2} = \sum_{k=1}^{M} \lambda_{\boldsymbol{m}_k} \varphi_{\boldsymbol{m}_k}(\boldsymbol{x}) \varphi_{\boldsymbol{m}_k}(\boldsymbol{z})$$

where $\boldsymbol{x}, \boldsymbol{z}$ are d dimensional vectors and $\boldsymbol{m}_k$ is a d-term multiindex stating the order of the eigenfunctions in each dimension. The $\mathbb{R}^d$ eigenfunctions and eigenvalues are defined as

$$\varphi_{\boldsymbol{m}_k}(\boldsymbol{x}) = \prod_{j=1}^{d} \varphi_{(\boldsymbol{m}_k)_j}((\boldsymbol{x})_j).$$

This was introduced in Section 4.3.2.

Given this small change in notation, all the previous definitions carry over naturally to higher dimensions; examples using this solution approach are discussed in Section 7.4. This flexibility in higher dimensions is one of the great benefits of working with meshfree kernel-based methods, but it does not necessarily mean that this is the optimal way of solving BVP in multiple spatial dimensions using Gaussian eigenfunctions. When presented with a suitably simple domain, it may be computationally efficient to choose points on a structured grid. This allows for 1D differentiation matrices to be combined to approximate higher dimensional differentiation matrices.

This idea was discussed in [168] for polynomial collocation, where it is especially useful because polynomial interpolation in 1D is better defined than in higher dimensions. In [60] this approach was extended to RBF-based collocation methods. The use of differentiation matrices for GaussQR approximation was developed in Chapter 6.

Two representative structured grids in 2D are displayed in Figure 7.4, although for this problem we consider only the Chebyshev tensor product grid in (x, y). These grids use N^2 points, with each "strip" of points containing N points. We can take advantage of the structure of these grids by noting that each vertical strip of points contains the same ordering of y values with the x value constant; this

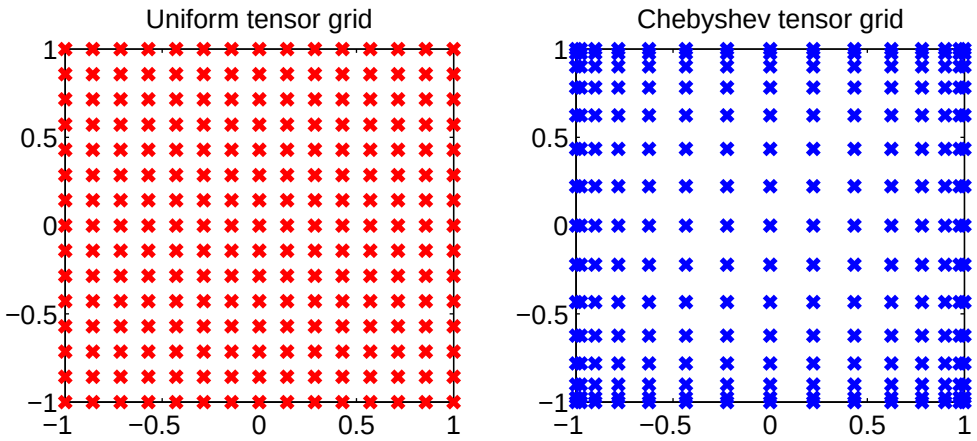


Figure 7.4: These 2D grids are actually structured copies of 1D grids. For any fixed x (or y) the distribution of y (or x) points is identical.

allows the same differentiation matrix to apply on each vertical strip. A similar statement can be made for each horizontal strip of points.

Assume that we have a differentiation matrix D which applies the differential operator $\mathcal{D}$ to a vector of values evaluated at $x_1, \ldots, x_N$. By placing the function values $u(x, y)$ in the vector $\boldsymbol{u}^T$ in the order

$$(\, u(x_1, y_1) \, \cdots \, u(x_1, y_N) \, u(x_2, y_1) \, \cdots \, u(x_2, y_N) \, \cdots \, \cdots u(x_N, y_1) \, \cdots \, u(x_N, y_N) \,),$$

the differentiation matrix D can be applied in the x direction on the 2D grid with the matrix vector product

$$(\mathsf{I}_N \otimes \mathsf{D})\boldsymbol{u}.$$

Here $\otimes$ represents the Kronecker tensor product [171]. We can obtain a similar result in the y direction with the product

$$(\mathsf{D} \otimes \mathsf{I}_N)\boldsymbol{u}.$$

If we were to construct a second derivative operator D on N 1D Chebyshev nodes, the Laplacian on the N^2 2D Chebyshev tensor grid would take the form

$$\mathsf{I}_N \otimes \mathsf{D} + \mathsf{D} \otimes \mathsf{I}_N.$$

By replacing rows associated with boundary values of (x, y) with the associated boundary operator, we may solve boundary value problems with this differentiation matrix approach. As an example, we solve the Helmholtz problem

$$\nabla^2 u(x,y) + \nu^2 u(x,y) = f(x,y), \qquad -1 < x < 1, -1 < y < 1 \tag{7.13a}$$

$$u(x,y) = g(x,y), \qquad |x| = 1 \cup |y| = 1 \tag{7.13b}$$

using $\nu = 7$. The true solution is chosen to be $u(x,y) = J_0(6\sqrt{x^2+y^2})$, which necessitates that $f(x,y) = 13J_0(6\sqrt{x^2+y^2})$. Results are compared with $N = 20$ between tensor grid differentiation matrix solutions computed using polynomials (labeled "Trefethen"), the standard Gaussian basis (labeled "Fasshauer") and the stable basis (labeled "GaussQR"). The error is plotted as a function of the shape parameter in Figure 7.5.

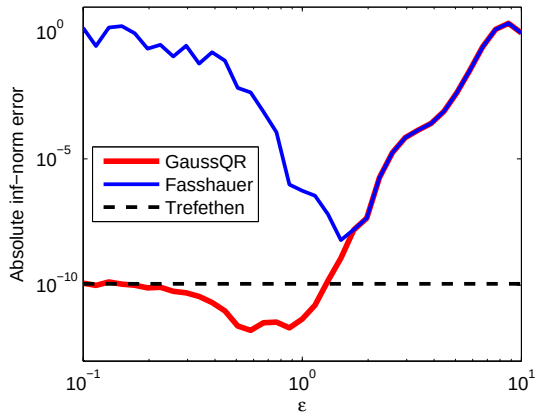


Figure 7.5: Polynomial (Trefethen), direct Gaussian (Fasshauer) and GaussQR differentiation matrices are tested for solving (7.13a). As was true for the 1D problems, the standard Gaussian collocation method fails in 2D for small ε, while the GaussQR method allows the solution to reach its $\varepsilon \to 0$ polynomial limit. $N^2 = 400$ collocation points are placed in the domain. For GaussQR, $\alpha = 1$ was used. The error is computed at the collocation points.

We can see here that, by using the Kronecker product on tensor style grids in multiple space dimensions, we can effectively implement the GaussQR method for BVP without being required to use GaussQRr, as was necessary for the interpo-

lation examples in Chapter 4. The ill-conditioning which would otherwise prevent this solution technique from its optimal accuracy is no longer a problem, and the computational cost is comparable to the polynomial collocation method. Because the differentiation matrix is only of size N, but the BVP linear system is of size N^2, there is significantly less cost in using RBF-QR to compute the differentiation matrix than in solving the full system.

7.4 The method of particular solutions using Gaussian eigenfunctions

When solving boundary value problems, it is often advantageous to transfer the problem to the boundary; the boundary is of lower dimension and requires less work to discretize, and irregularly shaped domains are less of a problem. The actual mechanism by which this is done can take multiple forms. Boundary element methods [87] (also called boundary integral methods [118, 10]) involve solving a related integral equation on the boundary, rather than a PDE on the domain.

Another approach, called the method of particular solutions (MPS), finds a function which satisfies the interior condition and then solves a simpler approximation problem only on the boundary. The solution on the interior is often called a particular solution, and it can be used in conjunction with the boundary element method to form the dual reciprocity method [132]. This section considers the applicability of the Gaussian eigenfunction expansion, and their associated stability for small ε, in finding particular solutions to boundary value problems.

7.4.1 The method of fundamental solutions

The method of fundamental solutions (MFS) is a powerful technique for solving homogeneous problems (i.e., with $f(\boldsymbol{x}) = 0$) with a linear operator $\mathcal{L}$ whose fundamental solution $G(x, z)$ is known. Its development is detailed in [55, 83]. We briefly cover some of that material here.

Essentially, MFS converts a boundary value problem to an interpolation problem. We assume that the problem of interest fits the form

$$\mathcal{L}u(x) = 0, \qquad x \in \Omega, \tag{7.14a}$$

$$\mathcal{B}u(x) = g, \qquad x \in \partial\Omega. \tag{7.14b}$$

The fundamental solution is a kernel which satisfies

$$\mathcal{L}G(x, z) = \delta(x, z),$$

where $\delta(x, z)$ is the Dirac delta function. We know that $\mathcal{L}G(x, z) = 0$ for $x \in \Omega$ if $z \notin \Omega$, because $\delta(x, z) = 0$ for $x \neq z$. The assumption is therefore made that the solution u is of the form

$$u(x) = \sum_{k=1}^{N} a_k G(x, z_k) \tag{7.15}$$

where the N kernel centers $\{z_k\}_{k=1}^N$ are placed outside $\Omega \cup \partial\Omega$.

Automatically, the condition (7.14a) is satisfied, meaning the coefficients $\{a_k\}_{k=1}^N$ must be determined by satisfying (7.14b). This is often accomplished by choosing N collocation points $\{x_k\}_{k=1}^N$ on the boundary, and then solving the linear system

$$\begin{pmatrix} \mathcal{B}G(x_1, z_1) & \cdots & \mathcal{B}G(x_1, z_N) \\ & \vdots & \\ \mathcal{B}G(x_N, z_1) & \cdots & \mathcal{B}G(x_N, z_N) \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} g(x_1) \\ \vdots \\ g(x_N) \end{pmatrix}.$$

It should be noted that the choice of N source terms is not required; often it is preferable to choose many fewer source terms than collocation points and solve an overdetermined system. Furthermore, the actual choice of source locations is sometimes also considered a variable in the problem. For simplicity, we only study problems with a fixed set of N sources.

In the simplest case, when $\mathcal{B} = \mathcal{I}$ (the Dirichlet boundary condition case), this is a kernel-based interpolation problem, using the basis $\{G(\cdot, z_k)\}_{k=1}^{N}$. More complicated boundary conditions are handled just as easily, and greater accuracy is expected than with a collocation method because of the absence of $\mathcal{L}$. Since $\mathcal{L}$ is a differential operator of higher degree than $\mathcal{B}$, more accuracy is lost when approximating it [176], making any solution involving both operators lower order than a solution involving only $\mathcal{B}$.

To demonstrate the impressive potential of the MFS, we apply it to the BVP

$$\begin{aligned} \nabla^2 u(x,y) &= 0, \qquad 0 \le x \le 1,\ 0 \le y \le \pi/2, \\ u(x,y) &= \mathrm{e}^x \cos(y), \qquad x = 0 \cup x = 1 \cup y = 0 \cup y = \pi/2. \end{aligned}$$

For comparison, we also solve this problem with the GaussQRr technique derived in Section 7.3.3, and a fourth order finite difference (FD) scheme [97]. The N MFS collocation points were chosen equally spaced on the boundary, and the source centers were equally spaced on the circle with radius 2 and center $x = .5$, $y = \pi/4$. The GaussQRr solution used parameters $M = .8N$, $\varepsilon = 10^{-8}$ and $\alpha = 1$, and placed half its collocation points on the 2D tensor product Chebyshev nodes and half on the Halton points [88]. This choice of points allows scattered data throughout the interior of the domain, and well-structured points on the boundary. The results are displayed in Figure 7.6.

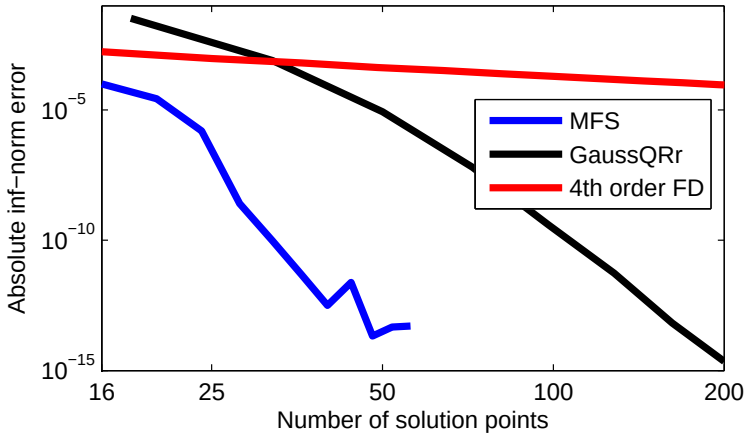


Figure 7.6: For the Laplace BVP, with Dirichlet boundary conditions, the MFS is vastly superior to finite differences, and even outperforms GaussQRr significantly.

It is clear that both the MFS and GaussQRr solutions are converging exponentially quickly, in contrast to the FD solution which is converging at its expected algebraic order. The MFS solution is much more accurate than the GaussQRr solution for fewer points. Part of this is the fact that GaussQRr places points on the interior and the boundary, and MFS only places points on the boundary because (7.14a) is satisfied analytically. Another factor contributing to the slightly worse behavior of GaussQRr is the presence of $\mathcal{L}$ in the system, requiring higher order derivatives which are approximated with less accuracy (recall Section 6.2.3). These factors combined suggest that for sufficiently simple problems, the method of fundamental solutions is still the king.

7.4.2 Finding particular solutions with GaussQRr

It is unsurprising that the method of fundamental solutions is more efficient than GaussQRr collocation, because it has the advantage of considering a solution only on the boundary. Unfortunately, the method of fundamental solutions is only ap-

plicable on homogeneous problems. To counteract this shortcoming, the method of particular solutions (MPS) was developed to allow for an inhomogeneous differential equation [128]. Recently, the method of particular solutions has been reconsidered and improved for solving eigenvalue problems on polygonal domains [16]; here we only consider MPS for boundary value problems.

In the MPS setting, the general BVP takes the form

$$\begin{aligned} \mathcal{L}u(\boldsymbol{x}) &= f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega, \\ \mathcal{B}u(\boldsymbol{x}) &= g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \partial\Omega, \end{aligned}$$

as was the case in Section 7.3; we assume, as we did in Section 7.4.1, that the operator $\mathcal{L}$ has the Green's function $G(x, z)$. For the MFS setting, $f \equiv 0$, meaning that the solution could be built with the basis $\{G(x, z_k)\}_{k=1}^{N_F}$, but now that $f \neq 0$, we assume the solution takes the form

$$u(\boldsymbol{x}) = u_F(\boldsymbol{x}) + u_P(\boldsymbol{x}).$$

The two components now solve different problems:

- $u_P(\boldsymbol{x})$ solves the ill-posed BVP $\mathcal{L}u_P(\boldsymbol{x}) = f(\boldsymbol{x})$. If collocation with the basis $\{K(x, z_k)\}_{k=1}^{N_P}$ is used to solve this problem, this can be thought of as an approximation problem on the interior, using the basis $\{\mathcal{L}K(x, z_k)\}_{k=1}^{N_P}$.
- $u_F(\boldsymbol{x})$ requires the particular solution, and solves the BVP

  $$\begin{aligned} \mathcal{L}u_F(\boldsymbol{x}) &= 0 && \boldsymbol{x} \in \Omega \\ \mathcal{B}u_F(\boldsymbol{x}) &= g(\boldsymbol{x}) - \mathcal{B}u_P(\boldsymbol{x}) && \boldsymbol{x} \in \partial\Omega \end{aligned}$$

  using MFS. This too is an approximation problem, only on the boundary, using the basis $\{G(x, z_k)\}_{k=1}^{N_F}$.

Because of the generally exceptional performance of the method of fundamental solutions, the main source of error for MPS is the approximation of the particular solution. This is a problem which may be remedied somewhat by the use of GaussQRr to find a particular solution, because one major source of error (the ill-conditioning for many values of ε) can be countered effectively. If we approximate the particular solution with Gaussian eigenfunctions,

$$u_P(\boldsymbol{x}) = \sum_{k=1}^{M} b_k \varphi_{\boldsymbol{m}_k}(\boldsymbol{x})$$

we can find the coefficients $\{b_k\}_{k=1}^{M}$ by choosing N_P points $\{\boldsymbol{x}_k\}_{k=1}^{N_P} \in \Omega$ and solving the approximation problem

$$\begin{pmatrix} \mathcal{L}\varphi_{\boldsymbol{m}_1}(\boldsymbol{x}_1) & \cdots & \mathcal{L}\varphi_{\boldsymbol{m}_M}(\boldsymbol{x}_1) \\ & \vdots & \\ \mathcal{L}\varphi_{\boldsymbol{m}_1}(\boldsymbol{x}_{N_P}) & \cdots & \mathcal{L}\varphi_{\boldsymbol{m}_M}(\boldsymbol{x}_{N_P}) \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_M \end{pmatrix} = \begin{pmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_{N_P}) \end{pmatrix}. \tag{7.16}$$

We can then determine the fundamental solution (of the form (7.15)) by choosing N_F collocation points $\{\hat{\boldsymbol{x}}_k\}_{k=1}^{N_F} \in \partial\Omega$, N_F source points $\{\boldsymbol{z}_k\}_{k=1}^{N_F} \notin \Omega \cup \partial\Omega$, and solving the linear system

$$\begin{pmatrix} \mathcal{B}G(\hat{\boldsymbol{x}}_1, \boldsymbol{z}_1) & \cdots & \mathcal{B}G(\hat{\boldsymbol{x}}_1, \boldsymbol{z}_{N_F}) \\ & \vdots & \\ \mathcal{B}G(\hat{\boldsymbol{x}}_{N_F}, \boldsymbol{z}_1) & \cdots & \mathcal{B}G(\hat{\boldsymbol{x}}_{N_F}, \boldsymbol{z}_{N_F}) \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_{N_F} \end{pmatrix} =$$
$$\begin{pmatrix} g(\hat{\boldsymbol{x}}_1) \\ \vdots \\ g(\hat{\boldsymbol{x}}_{N_F}) \end{pmatrix} - \begin{pmatrix} \varphi_{\boldsymbol{m}_1}(\hat{\boldsymbol{x}}_1) & \cdots & \varphi_{\boldsymbol{m}_M}(\hat{\boldsymbol{x}}_1) \\ & \vdots & \\ \varphi_{\boldsymbol{m}_1}(\hat{\boldsymbol{x}}_{N_F}) & \cdots & \varphi_{\boldsymbol{m}_M}(\hat{\boldsymbol{x}}_{N_F}) \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_M \end{pmatrix} \tag{7.17}$$

given the previously determined $\{b_k\}_{k=1}^{M}$.

To demonstrate the viability of this method, we apply it to the modified

Helmholtz problem

$$\nabla^2 u(x,y) - \nu^2 u(x,y) = f(x,y), \qquad -1 < x < 1,\ -1 < y < 1 \tag{7.18a}$$

$$u(x,y) = g(x,y), \qquad |x| = 1 \cup |y| = 1 \tag{7.18b}$$

using $\nu = 3$ and true solution $u(x,y) = \mathrm{e}^{x+y}$. The fundamental solution for the operator $\mathcal{L} = \nabla^2 - \nu^2 \mathcal{I}$ in $\mathbb{R}^2$ is

$$G(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{2\pi} K_0(\nu \|\boldsymbol{x} - \boldsymbol{z}\|),$$

where K_0 is the modified Bessel function of the second kind of order 0. For this example, we use $\nu = 3$, and compare the solution using GaussQRr approximate collocation to MPS using a GaussQRr generated particular solution.

The MPS solution uses N_F uniformly distributed points on the boundary for the MFS component, and $N_P \approx N_F$ Halton points on the interior for the GaussQRr particular solution. Source points are placed quasi-uniformly at a distance $\sim 1/\nu^2$ orthogonally away from the boundary. The GaussQRr collocation solution uses the same N_P points on the interior, and $N_B \approx .25 N_F$ points uniformly on the boundary. GaussQRr, for both the particular solution approximation and the collocation, uses the parameters $M = .5N_P$, $\varepsilon = 10^{-5}$ and $\alpha = 1$. For both methods, the error is computed at 35^2 points uniformly distributed throughout the domain. The results are displayed in Figure 7.7.

As we can see here, for $N_B < 140$ MPS is at least 10 times more accurate than GaussQRr, although the collocation technique does catch up soon after. Because $N_B \approx .25N_F$, $N_P \approx N_F$, and $M = .5N_P$, both methods have about the same cost:

- MPS has two costs: $\mathcal{O}(4/3 N_P (.5N_P)^2)$ for the least squares solve of the particular solution, and $\mathcal{O}(1/3 N_F^3)$. This total cost is roughly $\mathcal{O}(1/3(N_P^3 + N_F^3))$, or $\mathcal{O}(2/3 N_P^3)$.

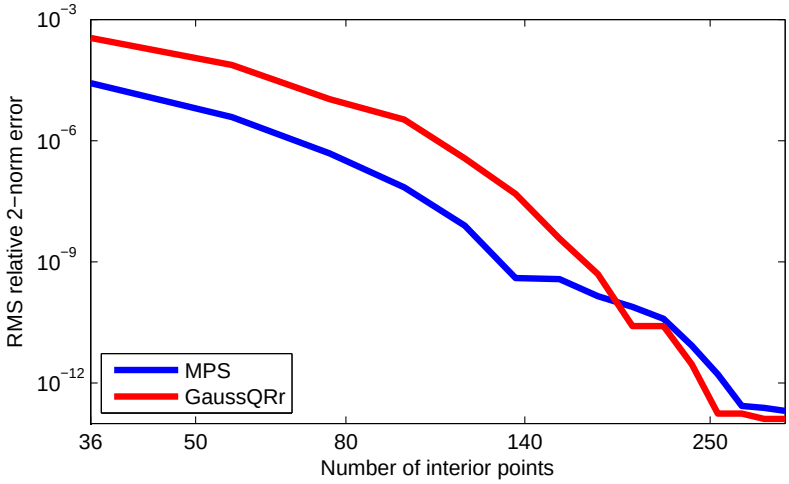


Figure 7.7: For the problem (7.18a), MPS using GaussQRr particular solution can be more effective than GaussQRr collocation. The x-axis is meant to represent the cost of the solve, because the cost in both settings is dominated by the interior solution.

- GaussQRr collocation requires a least squares solve of a system with $N_P + N_B$ rows and M columns. The cost of this is $\mathcal{O}(4/3(N_P + .25N_F)(.5N_P)^2)$ which is roughly $\mathcal{O}(5/12N_P^3)$.

This suggests that Gaussian eigenfunctions can be used to effectively approximate particular solutions, at least for problem as relatively simple as the one we have considered.

7.4.3 Incorporating collocation into the method of particular solutions

It was discussed in [176] that the accuracy of derivatives computed with an RBF interpolant are of a lower order than the interpolant itself; roughly one order of accuracy is lost per derivative taken. This was observed for GaussQRr approx-

imations in Section 6.2.3, and suggests that approximations generated with the basis $\{\mathcal{L}\varphi_{\boldsymbol{m}_k}\}_{k=1}^{M}$ are less accurate that those generated with the eigenfunction basis. Because of this, more complicated problems which require more accurate particular solutions may find MPS ineffective.

Collocation remains a viable option here, but it would be shameful to ignore the existence of the Green's functions given the excellent behavior of the method of fundamental solutions on homogeneous problems. Fortunately, it is not necessary to discard the MPS framework, because we can compute particular solutions using collocation. By incorporating boundary conditions into our particular solution, terms involving $\mathcal{B}\varphi_{\boldsymbol{m}_k}$ are included in the linear system, which benefits the accuracy because $\mathcal{B}$ is of lower order than $\mathcal{L}$.

This method differs slightly from the MPS described in Section 7.4.2.

- $u_P(\boldsymbol{x})$ solves the BVP

$$\begin{aligned} \mathcal{L}u_P(\boldsymbol{x}) &= f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega, \\ \mathcal{B}u_P(\boldsymbol{x}) &= g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \partial\Omega, \end{aligned}$$

  using $\{\boldsymbol{x}_k\}_{k=1}^{N_P} \in \Omega$ to handle the PDE and $\{\hat{\boldsymbol{x}}_k\}_{k=1}^{N_B} \in \partial\Omega$ to handle the BC.

- $u_F(\boldsymbol{x})$ requires the particular solution, and solves the BVP

$$\begin{aligned} \mathcal{L}u_F(\boldsymbol{x}) &= 0, \qquad \boldsymbol{x} \in \Omega, \\ \mathcal{B}u_F(\boldsymbol{x}) &= g(\boldsymbol{x}) - \mathcal{B}u_P(\boldsymbol{x}), \qquad \boldsymbol{x} \in \partial\Omega, \end{aligned}$$

  using MFS. This is still an approximation problem on the boundary using the basis $\{G(\boldsymbol{x}, \boldsymbol{z}_k)\}_{k=1}^{N_F}$ and the collocation points $\{\tilde{\boldsymbol{x}}_k{}_{k=1}^{N_F}\} \in \partial\Omega$.

The difference with the earlier MPS is that the points $\tilde{\boldsymbol{x}}_k$ must be chosen differently than the points $\hat{\boldsymbol{x}}_k$, i.e., $\tilde{\boldsymbol{x}}_k \neq \hat{\boldsymbol{x}}_j$ for $1 \leq k \leq N_F$ and $1 \leq j \leq N_B$. If the boundary

points were chosen the same for both the collocation and MFS, then the MFS would be tricked into believing $g(\boldsymbol{x}) - \mathcal{B}u_P(\boldsymbol{x}) = 0$ everywhere because the collocation would have already satisfied $g(\hat{\boldsymbol{x}}) = u_P(\hat{\boldsymbol{x}})$.

To test this method, we'll consider a more difficult problem than our previous MPS test. The BVP now has mixed boundary conditions

$$\nabla^2 u(x,y) - \nu^2 u(x,y) = f(x,y), \qquad (x,y) \in \Omega, \tag{7.19a}$$

$$u(x,y) = g_D(x,y), \qquad (x,y) \in \Gamma_D, \tag{7.19b}$$

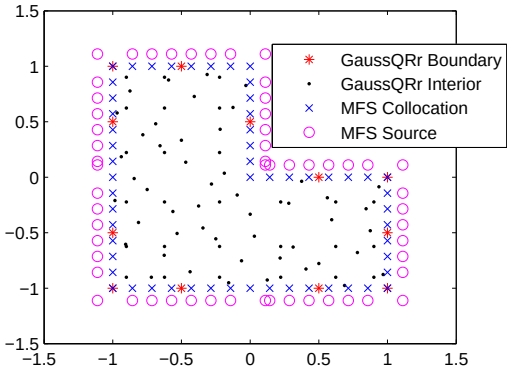
$$\frac{\partial}{\partial \boldsymbol{n}} u(x,y) = g_N(x,y), \qquad (x,y) \in \Gamma_N, \tag{7.19c}$$

on the L-shaped geometry

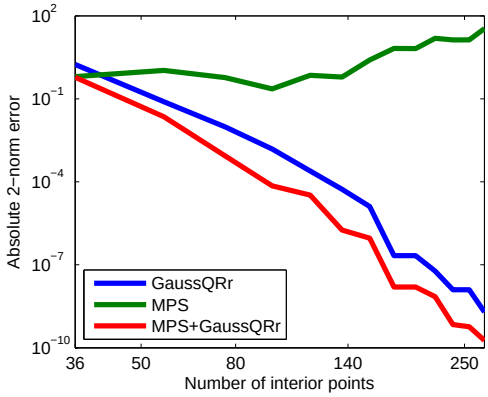
$$\begin{aligned}
\Omega &= \{x \in (-1,1), y \in (-1,1) \mid x < 0 \cup y < 0\}, \\
\Gamma_D &= \{x \in [-1,1], y \in [-1,1] \mid x = -1 \cup (x = 0 \cap y > 0) \cup (x > 0 \cap y = 0)\}, \\
\Gamma_N &= \{x \in [-1,1], y \in [-1,1] \mid y = -1\}.
\end{aligned}$$

The setup of the problem, and the solution results are found in Figure 7.8.

Figure 7.8a explains the distribution of collocation and source points chosen for the various solution methods. Three solution techniques are compared in Figure 7.8b: "MPS" uses GaussQRr interpolation on the interior to generate particular solutions and MFS to enforce the boundary; "GaussQRr" uses GaussQRr collocation from Section 7.3.3 to solve the full boundary value problem; "MPS+GaussQRr" uses the GaussQRr collocation solution as the particular solution and the MFS to enforce for the boundary terms. The "MPS+GaussQRr" solution is the most effective, and perhaps most noteworthy is that the quality of the particular solution is so much more accurate after incorporating only a small number of boundary terms. It can safely be assumed that the improvement comes in the particular solution



(a) GaussQRr uses N_P Halton points on the interior, and a roughly uniform set of $N_B \approx .2N_P$ points on the boundary. MFS uses $N_F \approx .7N_P$ points uniformly on the boundary and N_F points at a distance $\sim 1/\nu^2$ from the boundary. This sample point distribution was created with $N_P = 76$.



(b) The MPS using the interpolation particular solution falters almost immediately, whereas the GaussQRr solution converges similarly to the earlier example. The introduction of a small number of boundary terms to the particular solution allows for the "MPS+GaussQRr" solution to converge much better than the "MPS", and even better than "GaussQRr".

Figure 7.8: We have chosen the true solution $u(x, y) = \sin(x^2 + y)$, and modified Helmholtz parameter $\nu = 3$. For this example, the refinement step of performing MFS on the GaussQRr collocation solution provides as much as an extra order of accuracy. GaussQRr techniques used the parameters $M = .5N_P$, $\varepsilon = 10^{-6}$ and $\alpha = 1$.

because that is the only difference between the "MPS" and "MPS+GaussQRr" curves.

In some sense, by computing the particular solutions with collocation, we have now shifted the burden of the solution from primarily on the boundary to primarily on the interior. For the traditional MPS, the particular solution is not unique, and the actual solution itself is governed by the MFS component. In this slightly different setting, the solution is first computed with collocation, and then MFS is used as a refinement technique to more effectively incorporate the boundary. Research is needed to determine if the MFS refinement could have a detrimental effect on the final solution, but in this one example, it only helps.

The choice of boundary points seems especially relevant for this setup. Because the GaussQRr method is performing approximate collocation (because $M < N_P + N_B$), it is unlikely that $g(\hat{\boldsymbol{x}}_k) = u_P(\hat{\boldsymbol{x}}_k)$ and therefore even more unlikely that $g(\tilde{\boldsymbol{x}}_k) = u_P(\tilde{\boldsymbol{x}}_k)$. Even so, if the MFS is tricked into thinking that the particular solution is doing a very good job, when in fact it is only doing a good job on a select set of points, then the MFS will not be improving the solution as much as it could. No specific actions were taken here to ensure that the GaussQRr collocation and MFS shared no boundary points, although Figure 7.8a suggests that at least some of the points did not overlap. In the future, it may be possible to fix the source points $\{\boldsymbol{z}_k\}_{k=1}^{N_S}$ and adaptively choose the MFS points $\{\tilde{\boldsymbol{x}}_k\}_{k=1}^{N_F}$ to account for the locations which collocation least accurately solved by solving an overdetermined system.

7.5 Summary

We have presented methods, based on the GaussQR interpolation scheme, for solving boundary value problems. Collocation techniques, drawn from standard kernel-based collocation, proved useful for overcoming the traditional ill-conditioning associated with the flat RBF limit. The GaussQRr interpolation technique was also considered as a method for generating particular solutions within the Method of Particular Solutions. GaussQRr collocation proved even more useful for generating particular solutions, allowing for an accurate solution with a reasonable amount of work. Adding these tools to the multiphysics infrastructure provides BVP solvers which can solve appropriate systems with high levels of accuracy, and couple effortlessly to other simulations.

Looking ahead, we are interested in determining, for the collocation setting, the effect of adding a polynomial basis on the optimal ε value for the solution. In Section 7.3.1 we introduced the idea, but dropped it to focus on the GaussQR replacement of direct Gaussian collocation. Given that the $\varepsilon \to 0$ limit produces polynomials, it is not necessary to include a polynomial term in the approximation to reproduce a truly polynomial solution. Even so, if a polynomial term were present, it might change the optimal ε curve, and potentially also the optimal error that can be achieved. This research would be relatively straightforward to conduct, and would benefit many applications which already include polynomial terms in their solutions.

The same uncertainties which stymie the GaussQR technique in the interpolation setting are present in the solution of boundary value problems. Specifically, the free parameters ε, α and M need to be chosen correctly to take advantage of the potentially optimal accuracy available to kernel methods. Thus far, this work serves only as a proof of concept, and significant research needs to be done to provide good parameter values for general applications. Possible avenues for making informed parameter choices include extending existing statistical methods for determining ε (such as cross-validation and maximum likelihood estimation) to include α and M; this topic is addressed to some degree in Chapter 8. It may also be possible to study the parameter choices as N increases, and to run many experiments for smaller N to make a smarter decision for larger N. This lack of analytic support regarding parameter choices is one of the fundamental stumbling blocks for advances in kernel-based methods.

Computational cost is also of great significance to any practical application, and much work needs to be done to make these methods useful in a high per-

formance environment. The presence of dense matrices, as is often the case in kernel-based methods, is magnified by the need to perform a QR factorization for both GaussQR and GaussQRr. This is mitigated somewhat in the tensor grid setting discussed in Section 7.3.5, but for those sparse systems, appropriate iterative solvers [44] and preconditioning schemes need to be developed. Work has been done for general RBFs to incorporate tree-code [110] and FMM methods to allow for faster kernel evaluations (mentioned in Section 5.6), and it is likely that applying these methods to the GaussQR framework will improve the computational prospects. The preconditioning approach used in Section 6.3.2 may be a useful tool in this setting as well.

CHAPTER 8

STATISTICAL INFERENCE FOR CHOOSING EIGENFUNCTION PARAMETERS

8.1 Introduction

The work involving kernel-based methods in this thesis has been presented to solve, at least in some circumstances, the ill-conditioning issue associated with certain parameterizations of Gaussian kernels. Many of the solution curves we have produced (see e.g., Figure 4.1, Figure 6.3, Figure 7.1) indicate the existence of an optimal ε parameterization. Unfortunately, we have not suggested how these optimal ε values can be determined prior to computing the actual error in the interpolation.

The problem of optimal parameterization is one that plagues users of kernel methods. Some kernels, such as polyharmonic splines, circumvent the problem by choosing kernels without a shape parameter. For a few circumstances, we know the optimal value of ε; for instance, when trying to reproduce a polynomial using Gaussians, the optimal $\varepsilon = 0$. Most problems are not as simple, and to truly achieve the excellent performance expected of kernel-based approximation theory, we need to develop a method for determining ε which works well for general problems.

Some research has been done for determining shape parameters for specific applications (see [73, 158, 111, 166, 173]). Other research has tried to isolate a single kernel or type of kernel and consider the effect for general problems (see [74, 145, 65, 150]). Our goal here is not to solve this problem, but rather to discuss

the validity of existing methods in the context of the new stable basis.

The primary methods we consider here involve statistics, and as such we use Section 8.2 to introduce the scattered data interpolation problem from a Gaussian processes perspective. Section 8.3.1 covers the current statistical methods used to parameterize kernels, and studies how these methods function once the stable Gaussian basis is implemented. These methods involve a determinant evaluation, which can be difficult to compute in some circumstances, so a new technique for approximating determinants is developed and tested in Section 8.4. Section 8.5 combines some of the results from Section 8.3.1 and Section 8.4 to stably perform a stochastic simulation to approximate a distribution describing the likelihood of a given ε producing the data of the problem.

8.2 Kernel-based approximation through Gaussian processes

Thus far we have thought of kernel-based approximation as a deterministic technique. If we want to introduce stochastic error into our data, or assume that our given data has stochastic noise in it, we would benefit from rephrasing our problem in terms of Gaussian processes. For a thorough discussion about radial basis functions and Gaussian processes, see [163] or [137].

Suppose we want to fit the model

$$Y(\boldsymbol{x}) = Z(\boldsymbol{x}) + \sum_{k=1}^{N_f} \beta_k f_k(\boldsymbol{x}) \tag{8.1}$$

to data $(\boldsymbol{x}_k, y_k)$, $1 \leq k \leq n$; we assume here that the output values y_k are scalar, although more complicated settings are possible. The f_k terms are deterministic

functions - eg. they may be polynomials if our model expects polynomial reproduction. Z is a Gaussian Process with zero mean and covariance

$$\mathrm{Cov}(Z(\boldsymbol{u}), Z(\boldsymbol{v})) = \sigma^2 K(\boldsymbol{u}, \boldsymbol{v}), \tag{8.2}$$

where $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the covariance kernel. Compare the structure of this Y process to the collocation solution (7.1): both have kernel components added to other terms which are specific to the problem at hand.

We restrict $K(\boldsymbol{u}, \boldsymbol{v}) = \phi(\varepsilon\|\boldsymbol{u} - \boldsymbol{v}\|)$ to be a radial basis function, with ε the shape parameter. σ^2 is the process variance. Note that at times we refer to the points $\boldsymbol{x}_k$ as the design of the process or experiment, in keeping with the nomenclature from statistics literature. For simplicity, we ignore the regression terms by setting $\beta_j \equiv 0$ – this means we presume the data (x_i, y_i) is a realization of Z. This same choice was made in Section 7.3.1 so that we could focus only on the kernel component. Choosing to include the deterministic functions is useful in some applications, or when ϕ is chosen to be conditionally positive definite.

Given a shape parameter ε, the best linear unbiased predictor for a new $\hat{\boldsymbol{x}}$ is

$$\hat{y}(\varepsilon) = \boldsymbol{k}(\hat{\boldsymbol{x}})^T \mathsf{K}^{-1} \boldsymbol{y} \tag{8.3}$$

where $(\mathsf{K})_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the covariance matrix, $\boldsymbol{y}$ is a vector of the design values, and

$$\boldsymbol{k}(\boldsymbol{x})^T = (K(\boldsymbol{x}, \boldsymbol{x}_1), \cdots, K(\boldsymbol{x}, \boldsymbol{x}_N))$$

is the vector of covariances between $\boldsymbol{x}$ and the design points.

As discussed in Chapter 4, the choice of shape parameter ε greatly affects the accuracy of the resulting RBF interpolant. As an example, consider input data generated by evaluating $(1 + x^2)^{-1}$ at 6 evenly space points between 0 and 1. We

compare the unique polynomial interpolant of these points to the Gaussian RBF interpolant for various values of ε in Figure 8.1. Note here that the RBF-Direct method was used, rather than GaussQR, because the optimal value could be found for this very small problem without the eigenfunction expansion.

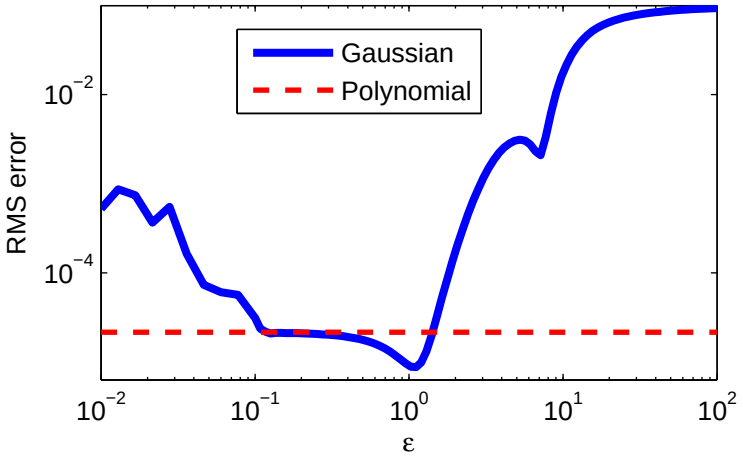


Figure 8.1: Near $\varepsilon = 1$, the Gaussian interpolant reaches its optimal accuracy, although we would not know that if we did not have the true solution. For this problem, $N = 6$ evenly spaced points on $[0, 1]$ are used as input, the function of interest is $f(x) = 1/(1 + x^2)$, and the error is tested at 100 evenly spaced points in the domain.

It is apparent in that graph that for some values of ε, the RBF interpolant outperforms the simple polynomial interpolation. Of course, finding the *best* ε is not a straightforward problem; even defining what *best* means is a subjective decision, and several methods for defining and finding a good ε value are discussed in Section 8.3.1. One definition which is not practical is the actual error associated with the interpolant, because many applications do not have a true solution with which to compare. Thus, despite the fact that we now "know" the optimal shape parameter for this example, this metric is not useful in general.

One way forward is to take advantage of our assumption that the data is generated by a Gaussian process. For all Gaussian processes, the likelihood of the

parameters given the data is

$$L(\varepsilon, \sigma^2) = p(\varepsilon, \sigma^2 | \boldsymbol{y}) \propto \frac{\exp\left(-\frac{1}{2\sigma^2}\boldsymbol{y}^T \mathsf{K}^{-1}\boldsymbol{y}\right)}{(\sigma^2)^{N/2}\sqrt{\det(\mathsf{K})}}.$$

See [130] for a derivation. Because of the special structure of this function, σ^2 can be analytically integrated out to find

$$M(\varepsilon) = \int_0^\infty L(\varepsilon, \sigma^2) d\sigma^2 = (\boldsymbol{y}^T \mathsf{K}^{-1}\boldsymbol{y})^{-N/2} \det(\mathsf{K})^{-1/2}, \tag{8.4}$$

which is the function that we want to maximize. Note that M can be interpreted as $p(\varepsilon|\boldsymbol{y})$, the profile likelihood over all σ^2. The process of maximizing that particular function is threatened by over/underflow, so instead we try to minimize a scaled version of its negative logarithm, which is

$$\tilde{M}(\varepsilon) = \log(\boldsymbol{y}^T \mathsf{K}^{-1}\boldsymbol{y}) + \frac{1}{N}\log\det(\mathsf{K}). \tag{8.5}$$

Each evaluation of $\tilde{M}$ requires a linear solve and a determinant evaluation. In Section 8.2.1, some of the properties of the determinant of a symmetric positive definite matrix are be discussed. In Section 8.2.2 we briefly discuss some iterative techniques used to solve ill-conditioned linear systems arising in RBF interpolation.

8.2.1 Determinant review

A standard definition of the determinant of a matrix A is

$$\det(\mathsf{A}) = \prod_{i=1}^{n} \lambda_i, \tag{8.6}$$

where λ_i is the i^{th} eigenvalue of A. This is not the only definition, but it suits our purposes. The standard technique for computing the determinant is by a decomposition of the matrix. For instance if you already have the Cholesky factorization

of $\mathsf{A} = \mathsf{L}\mathsf{L}^T$ then the determinant can be computed as

$$\det(\mathsf{A}) = \det(\mathsf{L})\det(\mathsf{L}^T) = \det(\mathsf{L})^2.$$

Because L is a triangular matrix its eigenvalues are all on its diagonal, so once you have the decomposition of A, computing its determinant is trivial. Error does exist in the Cholesky factorization process, so this does not suggest that the determinant can be found without any error. Rather we are expressing that if the matrix A has already be factored, that the determinant of the factors can be used to find the determinant of A. If however there is no stable way to factor the matrix, finding the determinant may be more troublesome.

Few modern numerical algorithms require the computation of the determinant because doing so is often more computationally expensive than other methods. As mentioned in Section 8.2, the most common appearance of determinants today are in the density function for the multivariate Normal distribution

$$L(\boldsymbol{x}; \boldsymbol{\mu}, \mathsf{K}) = \frac{\exp(-1/2(\boldsymbol{x} - \boldsymbol{\mu})^T \mathsf{K}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}))}{\sqrt{(2\pi)^N \det(\mathsf{K})}}.$$

Evaluating these likelihood functions, or the Kullback-Leibler divergence [46, 17] associated with Normal distributions, requires a determinant evaluation. Many statistical settings accompany the evaluation of $\det(\mathsf{A})$ by a linear solve $\mathsf{A}^{-1}\boldsymbol{b}$ - thus if a factorization is used to find $\mathsf{A}^{-1}\boldsymbol{b}$, $\det(\mathsf{A})$ is found at no additional cost. Conversely, if the linear solve is conducted in an iterative fashion there may be no determinant-revealing decomposition; in this case computing the determinant must be done separately. Ideally, finding the determinant in this case should have the same complexity as the linear solve.

8.2.2 Iterative methods for RBF problems

Any iterative linear solver for symmetric positive definite systems would theoretically be appropriate for the systems associated with positive definite RBF interpolants. The ill-conditioning problems discussed in Chapter 4 make many iterative methods impractical for solving RBF linear systems, both because of the lack of reliable preconditioning, and because the matrices may not be positive definite at machine precision.

More than 50 years ago, a regularization technique for solving ill-conditioned symmetric positive definite linear systems was developed in [139]. It resurfaced in [131] and while it does not have an official name, we call this Riley's Algorithm. We mention this algorithm to suggest that, while methods for solving ill-conditioned systems exist, they rarely involve the decomposition of the matrix.

Assuming the goal is to solve the system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$ where A is SPD and ill-conditioned, Riley's algorithm solves a regularized system involving the matrix

$$\mathsf{C} = \mathsf{A} + \mu \mathsf{I}_N \qquad \mu > 0 \tag{8.7}$$

which can be factored safely with the Cholesky decomposition. If all we concerned ourselves with was solving $\mathsf{C}\boldsymbol{y} = \boldsymbol{b}$ this would be ridge regression or Tikhonov regularization. But we can take another step by noting the identity

$$\mathsf{A}^{-1} = \frac{1}{\mu} \sum_{k=1}^{\infty} (\mu \mathsf{C}^{-1})^k,$$

which gives us a simple iteration method for approximating the solution to $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$

$$\boldsymbol{x}_{i+1} = \boldsymbol{y} + \mu \mathsf{C}^{-1} \boldsymbol{x}_i \tag{8.8}$$

where $\boldsymbol{y} = \mathsf{C}^{-1}\boldsymbol{b}$. This technique allows us to find $\boldsymbol{x}$ by only performing a stable Cholesky decomposition on C.

Choosing the regularization parameter μ to maximize stability but minimize the summation length is not a straightforward procedure, but we won't concern ourselves with it here. What we do need to be concerned with is that evaluating (8.5) requires both $\mathsf{K}^{-1}\boldsymbol{y}$ and $\det(\mathsf{K})$. Unfortunately, using Riley's algorithm for the linear solve does not provide us with a determinant revealing factorization, meaning that the determinant must be computed separately.

A more recent iterative solver for ill-conditioned and symmetric positive semi-definite linear systems is MINRES-QLP [44]. This technique allows for more stable solution to the system which previously was subject to unexpected results when using CG and MINRES. Given the existence now of a best M-term approximation to the Gaussian using (4.8), it is possible to perform fast matrix-vector products involving K, which makes MINRES-QLP a viable option. Work on this with Sou-Cheng Choi is in progress, but it is mentioned here to indicate that iterative solvers for RBF systems are more practical than they previously were.

8.3 Kernel parameterization

As described in Section 8.2, there is a significant difference between knowing that an "optimal" ε value exists, and being able to find it. In Figure 8.1, as well as figures from Chapter 4 and Chapter 7, it is apparent that a good choice provides significant advantages, and a poor choice adversely affects the solution accuracy. We briefly introduced the use of maximum likelihood estimation, and this section explains that method, along with other methods which have been developed for finding an optimal ε. After describing these methods, we study the impact of the eigenfunction expansion in allowing for their stable evaluation.

8.3.1 Existing methods for kernel parameterization

Members of the RBF and kernel-based approximation community have struggled for years to sufficiently define and determine optimal shape parameters for their approximation and collocation problems. At times, the shape parameter has been chosen to balance the error saturation phenomenon discussed in [23]. The kriging variance, or power function [60], can be used to guide the choice of ε because of its presence in the native space error estimates; the power function evaluated at a point $\boldsymbol{x}$ is

$$P_{K,\chi}(\boldsymbol{x}) = \sqrt{K(\boldsymbol{x},\boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^T \mathsf{K}^{-1} \boldsymbol{k}(\boldsymbol{x})},$$

where K comes from the interpolation system $\mathsf{K}\boldsymbol{c} = \boldsymbol{y}$, $\boldsymbol{k}(\boldsymbol{x})$ was defined in (4.13), and $P_{K,\chi}$ denotes the fact that the function depends on the design $\chi = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ and the kernel K. Because solutions using the standard basis incur severe ill-conditioning for $\varepsilon \to 0$, sometimes the ε is chosen as part of a trade-off principle [151]: pick ε as small as possible such that the system can still be solved with some accuracy. In [113], the authors studied the optimal ε values as a function of N, suggesting that tests run with small N can produce better ε guesses for large N, where such tests are impractical. This approach also proved useful in Section 6.3.1, where optimal ε ranges for small multiphysics simulations suggested the optimal ranges for larger problems.

There is another branch of techniques based on the statistical interpretation of the problem through Gaussian processes rather than the approximation theoretic approach we have considered before this chapter. These techniques make assumptions about the underlying nature of the problem (essentially that the data we see is a realization of the Gaussian process defined in (8.1)) and proceed to optimize some statistic involving ε based on that assumption. Cross-validation

[4] is one such technique, where a learning set is chosen to create an interpolant, whose error on the validation set is used to calibrate the choice of ε. This approach gained popularity after leave-one-out cross validation (LOOCV) was proven to be of manageable cost when implemented as described in [140].

Another technique involving statistics was suggested earlier in Section 8.2 while we were introducing the concept of Gaussian processes. The likelihood of the covariance kernel with shape parameter ε having generated the data $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N)\}$ can be computed using (8.4). Therefore, by maximizing that likelihood function, we choose the ε which is most likely to have generated that data, assuming all our other assumptions were true. This likelihood function is incorporated into the posterior distribution in Section 8.5, to allow us to perform more elaborate studies of the effect of ε.

8.3.2 Using eigenfunctions within parameterization schemes

In Chapter 4 we showed that the ill-conditioning present in Gaussian interpolation can be circumvented by rephrasing the problem in a stable basis derived from the eigenfunction expansion described in Section 4.3. The ε parameterization strategies of the previous section need to be reevaluated within the context of the stable basis to determine which methods are feasible.

For instance, the power function requires the quadratic term $\boldsymbol{k}(\boldsymbol{x})^T \mathsf{K}^{-1} \boldsymbol{k}(\boldsymbol{x})$,

which we could rewrite using the stable basis as

$$\begin{aligned}\boldsymbol{k}(\boldsymbol{x})^T\mathsf{K}^{-1}\boldsymbol{k}(\boldsymbol{x}) &= \boldsymbol{k}(\boldsymbol{x})^T(\Psi\Lambda_1\Phi_1^T)^{-1}\boldsymbol{k}(\boldsymbol{x}) \\ &= \boldsymbol{k}(\boldsymbol{x})^T\Phi_1^{-T}\Lambda_1^{-1}\Psi^{-1}\boldsymbol{k}(\boldsymbol{x}),\end{aligned}$$

where we have assumed the truncation point $M > N$. Here, $\mathsf{K} = \Psi\Lambda_1\Phi_1$ can be thought of as a sort of "Hilbert-Schmidt SVD", where Φ_1 and Ψ are populated by the functions related to the Hilbert-Schmidt eigenfunctions, and Λ_1 is a diagonal matrix populated by the first N Hilbert-Schmidt eigenvalues. Recall that these matrices were defined in Section 4.4.

We can assume, after properly choosing α and scaling the data, that computing $\boldsymbol{k}(\boldsymbol{x})^T\Phi_1^{-T}$ and $\Psi^{-1}\boldsymbol{k}(\boldsymbol{x})$ can be done stably; were this not the case, then the $\boldsymbol{\psi}$ basis would not produce the stable results seen earlier. We cannot, however, assume that Λ_1 can be inverted stably, despite the fact that it is a diagonal matrix with entries which can be inverted analytically.

The problem is not in computing the inverse, but rather in applying it to a vector such as $\Psi^{-1}\boldsymbol{k}(\boldsymbol{x})$. Because that vector is computed through a linear solve, it has terms bounded from below on the order of ϵ_{mach}. The matrix Λ_1 has no such restrictions, which causes $\Lambda_1^{-1}\left[\Psi^{-1}\boldsymbol{k}(\boldsymbol{x})\right]$ to have entries which are arbitrarily large as $\varepsilon \to 0$. In exact arithmetic, the terms in $\Psi^{-1}\boldsymbol{k}(\boldsymbol{x})$ would continue to shrink with those of Λ_1, but because of roundoff error, this is impossible. This is problematic because, in exact arithmetic, $0 \leq \boldsymbol{k}(\boldsymbol{x})^T\mathsf{K}^{-1}\boldsymbol{k}(\boldsymbol{x}) \leq 1$, but this computed result may be greater than 1, producing a complex (and therefore meaningless) kriging variance. A similar issue arose in Section 5.4.3 where ill-conditioning prevented the stable solution using the fast QR solver.

This result demonstrates that the stable basis may not provide a stable method

for evaluating the kriging variance as $\varepsilon \to 0$. On the opposite side of the spectrum, the so-called tradeoff principle is no longer relevant in the context of the stable basis, because we no longer need to worry about reaching the conditioning limit for the interpolation. There may still be some application for the conditioning tradeoff to determine α, but no longer for ε.

For the cross-validation option, the existence of the stable basis both opens and closes doors. On the one hand, the interpolant can now be computed stably for all values of ε, meaning that cross-validation is now applicable in regions where it was previously unreliable. Conversely, because we are no longer inverting the symmetric positive definite matrix K to compute the interpolant, we can no longer use Rippa's trick [140] to conduct the LOOCV in $\mathcal{O}(N^3)$ operations. Without a similar, and as yet unknown, trick in the stable basis, LOOCV would be prohibitively expensive.

One alternative would be to consider some other form of cross-validation which requires fewer linear solves. If 50%, or 67%, of the data were used as the training set then only a handful of systems would need to be solved to test each ε. While perhaps not as systematic as LOOCV, this technique would still provide some statistical support for choosing one ε value over another.

Figure 8.2 demonstrates the viability of cross-validation for the stable basis; the function $f(x) = \sin(2\pi x)$ is considered on $x \in [-1, 1]$ using $N = 24$ evenly spaced points. GaussQR uses $\alpha = 2$ and the training sets are chosen to be as uniformly distributed as possible, rather than the standard random approach. While none of the cross-validation methods is perfect at predicting the optimal ε, the two methods using the stable basis are more closely centered around it. LOOCV suffers from ill-conditioning too early and makes unreliable predictions in the optimal ε region.

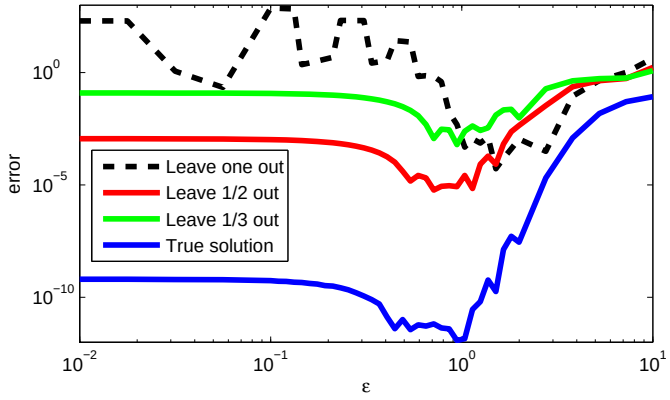


Figure 8.2: Leave half out and leave 1/3 out cross-validation methods using GaussQR are compared to LOOCV. The LOOCV suffers from the standard ill-conditioning, whereas the cross-validation methods using the stable basis predict with some accuracy where the optimal ε region is.

The final topic to consider here is the application of the eigenfunction expansion to maximum likelihood estimation. First, recall the negative log-likelihood function

$$\tilde{M}(\varepsilon) = \log(\boldsymbol{y}^T \mathsf{K}^{-1} \boldsymbol{y}) + \frac{1}{N} \log\det(\mathsf{K}) \tag{8.5}$$

which we want to maximize. The determinant term is tricky, because it generally requires a decomposition of K, which is dangerous in the stable basis. We could choose to use the Hilbert-Schmidt SVD described earlier to instead compute

$$\begin{aligned} \log\det(\mathsf{K}) &= \log\det(\Psi \Lambda_1 \Phi_1^T) \\ &= \log\Big[\det(\Psi)\det(\Lambda_1)\det(\Phi_1)\Big]. \end{aligned}$$

This is safer to compute because it involves two stable matrices, and we already have the factorization $\Phi_1 = \mathsf{Q}\mathsf{R}_1$ which was computed while forming Ψ. The determinant of Λ_1 can be determined analytically, so only $\det(\Psi)$ requires a new factorization.

While this seems like an ideal result, it does force us to use the GaussQR formulation to compute $\log\det(\mathsf{K})$. If we were instead interested in using the GaussQRr

(low rank $M < N$ series) formulation, we would be left without the Hilbert-Schmidt SVD. Because of this, we develop a statistical method for approximating $\log\det(\mathsf{K})$ in Section 8.4 when the low rank series is preferred.

Computing $\log(\boldsymbol{y}^T\mathsf{K}^{-1}\boldsymbol{y})$ incurs similar problems as the power function did, because applying the Hilbert-Schmidt SVD,

$$\log(\boldsymbol{y}^T\mathsf{K}^{-1}\boldsymbol{y}) = \log((\Phi_1^{-1}\boldsymbol{y})^T\Lambda_1^{-1}(\Psi^{-1}\boldsymbol{y})).$$

yields the same ill-conditioning in the Λ_1^{-1} term. Because there are no purely mathematical restrictions on this value, unlike the power function, we may be able to at least approximate this value to some accuracy.

In the same way that the truncated SVD of a matrix can be used to regularize the linear system, we could also consider the truncated Hilbert-Schmidt SVD to approximate K^{-1}. We would replace Λ_1^{-1} with Λ_1^{+}, where

$$(\Lambda_1^{+})_{k,k} = \begin{cases} (\Lambda_1^{-1})_{k,k} & \text{if } (\Lambda_1)_{k,k} > \tau \\ 0 & \text{else,} \end{cases}$$

for some tolerance τ. In doing this, the product $\Phi_1^{-T}\Lambda_1^{+}\Psi^{-1}$ would no longer be symmetric positive definite, as is guaranteed in exact arithmetic. By truncating early though, we gain a more stable method to compute K^{-1} to at least some accuracy.

When computing $\tilde{M}$ using this truncated Hilbert-Schmidt SVD, it may be necessary to likewise truncate the determinant computation. If $\log(\boldsymbol{y}^T\mathsf{K}^{-1}\boldsymbol{y})$ were truncated and $-\frac{1}{N}\log\det(\mathsf{K})$ were allowed to grow ever larger, the balance between the terms may be compromised. See Figure 8.3 for the likelihood as computed using both the Hilbert-Schmidt SVD and the truncated version. The problem is the same as was considered for cross-validation, and the figure shows that the

truncated MLE more closely follows the limiting behavior of the interpolant as $\varepsilon \to 0$. This likelihood function evaluation arises again in Section 8.5.

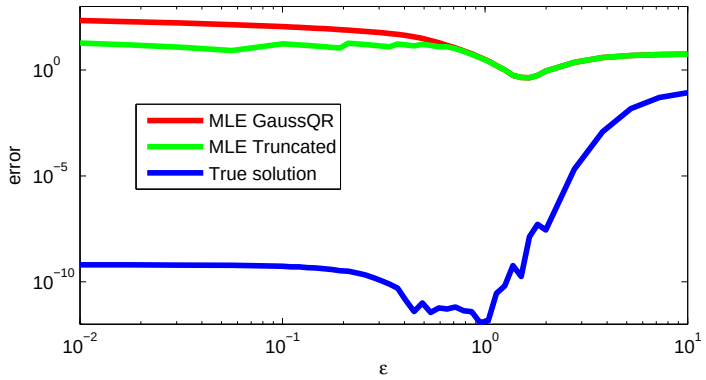


Figure 8.3: While not a perfect predictor, the log-likelihood function is a useful tool for helping choose a good ε. Note that the MLE GaussQR, without truncation, continues to increase as $\varepsilon \to 0$, whereas the truncated version (and the true solution) reach a limit. The tolerance $\tau = 10^{-14}$.

8.4 Approximating the determinant

As mentioned in Section 8.2.1, the determinant of the covariance matrix K needs to be computed in order to evaluate the log likelihood function (8.5). When a Cholesky decomposition is used to compute $\mathsf{K}^{-1}\boldsymbol{y}$, the determinant can be computed using the factors. If the solution is found iteratively using a technique from Section 8.2.2 (or some other method), no such factors exist, and the determinant must be computed separately. This section introduces an algorithm which approximates the derivative of symmetric positive definite systems.

This algorithm is first described in [11] for a matrix $\mathsf{A} = \mathsf{I}_n - \alpha\mathsf{D}$. D is a matrix whose eigenvalues are within $(-1, 1)$ and $\alpha \in (-1, 1)$ is a parameter in the model they were studying. For this case, the algorithm as follows approximates the log-determinant of A.

Algorithm 6 Barry/Pace log-determinant algorithm

Given $\mathsf{D} \in \mathbb{R}^{n \times n}$, α
Choose $m, p > 0$
for k from 1 to p **do**
  $\boldsymbol{x} \leftarrow N(0, \mathsf{I}_n)$
  $\boldsymbol{v}_k \leftarrow -n \sum_{k=1}^{m} \boldsymbol{x}^T \mathsf{D}^k \boldsymbol{x} / \boldsymbol{x}^T \boldsymbol{x} \frac{\alpha^k}{k}$
end for
return mean($\boldsymbol{v}$)

At the end of this algorithm, we have a p-vector $\boldsymbol{v}$ and we can generate a 95% confidence interval for the true determinant $[\bar{\boldsymbol{v}} - F, \bar{\boldsymbol{v}} + F]$ where $\bar{\boldsymbol{v}} = 1/p \sum_k \boldsymbol{v}_k$ and

$$F = \frac{n\alpha^{m-1}}{(m+1)(1-\alpha)} + 1.96\sqrt{\frac{s^2(\boldsymbol{v})}{p}}. \tag{8.9}$$

$s^2(\boldsymbol{v})$ is the sample variance.

This algorithm is a direct extension from the power series representation

$$\log(1-x) = -\sum_{j=1}^{\infty} \frac{x^j}{j} \qquad x \in [-1, 1)$$

where we use the Rayleigh quotient to extract the eigenvalues. A proof of convergence is provided in [11] but for a basic understanding of why this works, we can think about the Schur decomposition of D

$$\mathsf{D} = \mathsf{Q}\Lambda\mathsf{Q}^T = \begin{pmatrix} & & \\ \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_n \\ & & \end{pmatrix} \begin{pmatrix} 1-\lambda_1 & & \\ & \ddots & \\ & & 1-\lambda_n \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_1^T \\ \vdots \\ \boldsymbol{q}_n^T \end{pmatrix},$$

where Q is orthonormal, and Λ is diagonal with ith value $1-\lambda_i$. When we compute $\boldsymbol{x}^T\mathsf{D}\boldsymbol{x}$ we are indirectly computing $\boldsymbol{y} = \mathsf{Q}^T\boldsymbol{x}$ which has the same distribution because Q is an orthogonal matrix. Because $\boldsymbol{y} \sim N(0, \mathsf{I}_n)$, then

$$\boldsymbol{y}^T\Lambda\boldsymbol{y} = \sum_{i=1}^{n} \boldsymbol{y}_i^2(1-\lambda_i), \tag{8.10}$$

so each inner product yields a weighted sum of the eigenvalues. Taking the quotient $\boldsymbol{x}^T\mathsf{D}\boldsymbol{x}/\boldsymbol{x}^T\boldsymbol{x}$ turns the coefficients in that summation into random variables with properties that allow us to derive (8.9). $\mathsf{Q}^T\boldsymbol{x}$ can be thought of as the vector whose i^{th} value is the *amount* of $\boldsymbol{x}$ which points in the direction of λ_i. If we choose enough $\boldsymbol{x}$ vectors which point randomly in all the directions then we should hit all the eigenvalues equally.

8.4.1 Extending the determinant algorithm for RBF matrices

Now that we have a technique which works on special matrices (for spectral radius $\rho(\mathsf{D}) < 1$ we need the eigenvalues of A between 0 and 2), what can we do to extend its usefulness to more general matrices?

- How can we use this algorithm on A whose eigenvalues are not in the acceptable range?
- Can we test for convergence before terminating the algorithm?
- Are there changes for this to be implemented efficiently in a vectorized setting (such as Matlab)?

While we are going through these adaptations, keep in mind the following thought: *The closer the eigenvalues are to* ± 1*, the slower the convergence.* This is a direct analog to the fact that the log series converges very slowly when the argument is far from the center, in that case 0.

Scaling the matrix

Suppose we know the largest eigenvalue of A is λ_1: that means that the largest eigenvalue of A/λ_1 is 1. If we call $\mathsf{D} = \mathsf{I}_n - \mathsf{A}/\lambda_1$,

$$\begin{aligned}\log\det(\mathsf{A}) &= \log\det(\lambda_1(\mathsf{A}/\lambda_1)) \\ &= n\log\lambda_1 + \log\det(\mathsf{A}/\lambda_1) \\ &= n\log\lambda_1 + \log\det(\mathsf{I}_n - \mathsf{D})\end{aligned}$$

So now we are able to use the algorithm on D and add $n\log\lambda_1$ to the result. This is the most obvious way to adapt A to an acceptable matrix. It has the disadvantage of scaling all the eigenvalues of the matrix by λ_1. If all the eigenvalues are near λ_1 then this is desirable.

Unfortunately if there is only one eigenvalue of A outside the region, then the other eigenvalues are scaled unnecessarily. This is unacceptable if the other eigenvalues of the matrix are close to zero, because scaling those brings the associated eigenvalues of D closer to the edge of convergence. If those eigenvalues were already slowing up the series convergence, they are even more troublesome now.

Consider the following RBF example: we want to use Gaussians to interpolate with $\varepsilon = .1$ and centers $x = 0 : .2 : 1$. The resulting matrix looks like

$$A = \begin{pmatrix} 1.0000 & 0.9996 & 0.9984 & 0.9964 & 0.9936 & 0.9900 \\ 0.9996 & 1.0000 & 0.9996 & 0.9984 & 0.9964 & 0.9936 \\ 0.9984 & 0.9996 & 1.0000 & 0.9996 & 0.9984 & 0.9964 \\ 0.9964 & 0.9984 & 0.9996 & 1.0000 & 0.9996 & 0.9984 \\ 0.9936 & 0.9964 & 0.9984 & 0.9996 & 1.0000 & 0.9996 \\ 0.9900 & 0.9936 & 0.9964 & 0.9984 & 0.9996 & 1.0000 \end{pmatrix}$$

and its eigenvalues are

$$6.0 \times 10^{0}$$
$$1.4 \times 10^{-2}$$
$$1.2 \times 10^{-5}$$
$$5.5 \times 10^{-9}$$
$$1.4 \times 10^{-12}$$
$$2.6 \times 10^{-16}$$

which is just dismal. The confidence interval for this matrix is still $[\bar{\boldsymbol{v}} - F, \bar{\boldsymbol{v}} + F]$, but F is now

$$F = \frac{n(1-\gamma)^{m-1}}{(m+1)\gamma} + 1.96\sqrt{\frac{s^2(\boldsymbol{v})}{p}}. \tag{8.11}$$

where $\gamma = \lambda_n/\lambda_1$. Even for this 6×6 problem, it takes a huge m to counteract the incredibly small γ. After $m = 1000000$ there still is not convergence - there would be no convergence even if the scaling were unnecessary, but the division by $\lambda_1 \approx 6$ still hurts.

While scaling allows the problem to be handled by the algorithm, it is only appropriate for a matrix whose eigenvalues are on the same order. Of course if a matrix has eigenvalues which are near each other but are not near zero, there is no ill-conditioning so the determinant can be found by a factorization. This seems to make scaling the matrix an ineffective technique; we will revisit it later in an acceleration framework.

Orthogonal draws

The problem with the tactic of scaling the spectrum to $(-1, 1)$ is that the entire system is punished for the delinquency of a small subset. If possible, it would seem

more logical to handle the offending eigenvalues separately so as to not slow down the convergence of the series by scaling. One way to do this is by a technique called orthogonal draws.

Suppose we know that there are only $n_\ell \ll n$ eigenvalues of $\mathsf{D} = \mathsf{I}_n - \mathsf{A}$ which are less than -1. If we had the orthogonal invariant subspace W of those n_ℓ eigenvalues (in $\mathsf{\Lambda}_\ell$), we could ensure that the draws x are from $\mathsf{W}^\perp$. This can be seen quickly by referring to (8.10) and noting that if $\boldsymbol{x} \in \text{span}(\mathsf{W}\ \perp)$ the $\boldsymbol{y}_i^2$ associated with the problematic eigenvalues will be 0. To visualize this, let's return to the Schur decomposition of D

$$\mathsf{D} = \mathsf{Q\Lambda Q}^T = \begin{pmatrix} & \\ \mathsf{W} & \mathsf{W}^\perp \\ & \end{pmatrix} \begin{pmatrix} \mathsf{\Lambda}_\ell & \\ & \mathsf{\Lambda}_- \end{pmatrix} \begin{pmatrix} \mathsf{W}^T & \\ & \left(\mathsf{W}^\perp\right)^T \end{pmatrix}.$$

Now we define a symmetric projection matrix

$$\mathsf{P} = \mathsf{I}_n - \mathsf{W}\mathsf{W}^T$$

and consider

$$\begin{aligned} \mathsf{PDP} &= (\mathsf{I}_n - \mathsf{W}\mathsf{W}^T)\mathsf{V\Lambda V}^T(\mathsf{I}_n - \mathsf{W}\mathsf{W}^T) \\ &= \begin{pmatrix} & \\ 0 & \mathsf{W}^\perp \\ & \end{pmatrix} \begin{pmatrix} \mathsf{\Lambda}_\ell & \\ & \mathsf{\Lambda}_- \end{pmatrix} \begin{pmatrix} 0 & \\ & \left(\mathsf{W}^\perp\right)^T \end{pmatrix}. \end{aligned}$$

From this it is obvious that the only nonzero eigenvalues of matrix PDP are in $\mathsf{\Lambda}_-$ which are within the acceptable range to be used in the algorithm. This means that we do not have to worry about drawing $\boldsymbol{x}$ from $\mathsf{W}^\perp$ because we can form PDP and use it in the algorithm with standard $\boldsymbol{x} \sim N(0, \mathsf{I}_n)$. The output

will be $\log\det(\Lambda_-)$, to which we will have to add $\log\det(\Lambda_\ell)$, determined when W was found.

Finding W can be done with a simple subspace iteration or with the Lanczos procedure. Of course, we will not know beforehand how many eigenvalues of D are less than -1. As a result, we will need to make a guess of n_ℓ and if the smallest eigenvalue associated with W is less than -1, a larger subspace will need to be found until W satisfies the requirements. Once an appropriate W is found, Computing PDP can be done as a preprocessing step of $O(n_\ell n^2)$ and then each multiplication requires only $O(n^2)$, or each time $\mathsf{PDP}\boldsymbol{x}$ needs to be computed it can be done in 3 matrix-vector products with total complexity $O(n^2) + O(n_\ell n)$.

Another aspect to consider is how we can combine scaling with orthogonal draws to speed up convergence. Let's look at the RBF generated matrix with $\varepsilon = .01$ for

$$K(x_i, x_j) = (1 - \varepsilon(\|x_i - x_j\|))_+ \tag{8.12}$$

in 1D with 30 evenly spaced points between 0 and 1. See [60] for a proof that this matrix is SPD. The spectrum that Matlab finds for this matrix is displayed as a histogram in Figure 8.4.

Because only one eigenvalue of K is greater than 1 (and thus only 1 eigenvalue of D less than -1) we only need to find the subspace of that one eigenvalue. If, however, we found the invariant subspace of the 5 largest eigenvalues, the only eigenvalues remaining are within $(.01, .00005)$. Using the algorithm on this matrix will converge in a reasonable time, but using the algorithm on a matrix K whose eigenvalues are within $(1, .005)$ will converge even more quickly.

Luckily, we have a technique for scaling the eigenvalues of a matrix; although

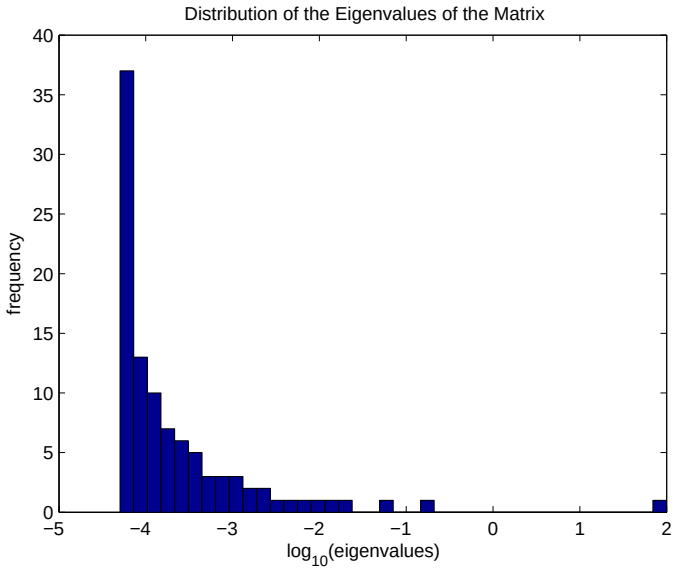


Figure 8.4: Only 1 eigenvalue of K is greater than 1

previously we used it to reduce the size of the spectrum. Consider the following idea to find $\det(\mathsf{K})$:

1. Use subspace iteration to find the ℓ largest eigenvectors/eigenvalues $\mathsf{W}/\mathsf{\Lambda}_\ell$. Because K is SPD, $\mathsf{W}^T\mathsf{K}\mathsf{W} = \mathsf{\Lambda}_\ell$.
2. Define (but don't form) $\mathsf{P} = \mathsf{I}_n - \mathsf{W}\mathsf{W}^T$, and call $\lambda = \min(\mathsf{\Lambda}_\ell)$.
3. Apply the algorithm to the matrix $\mathsf{P}(\mathsf{I}_n - \mathsf{K}/\lambda)\mathsf{P}$ to get ψ_1.
4. Compute our approximation $\log\det(\mathsf{K}) \approx \psi_1 + n\log\lambda + \text{sum}\log\text{diag}(\mathsf{\Lambda}_\ell)$.

This combination of orthogonal draws and scaling will allow us to approximate the determinant of K with a smaller log series summation m. This faster convergence is at the cost of performing the subspace iterations and the additional P multiplications. Because there is likely no way to know how many eigenvalues of K beyond the range of convergence, there needs to be flexibility built into the steps taken; however, with certain RBF we can expect that the number of offending eigenvalues is small.

See [176] and [60] for some discussion of the spectra for RBF interpolation matrices. Also, [22] discusses how the choice of design points $\boldsymbol{x}_i$ affects the determinant of the resulting matrix.

8.4.2 A Matlab version of log-determinant estimation

Before we look at all the changes discussed, let's look at the Matlab code used in [11].

```
function mclogdet = DetAppx(D,p,m)
n = length(D);v = zeros(p,1);
for i = 1:p
    x = randn(n,1);
    c = x;
    for k = 1:m
        c = D*c;
        v(i) = (x'*c)/k+v(i);
    end
    v(i) = -n*v(i)/(x'*x);
end
mclogdet = mean(v);
```

This code can easily be accelerated (more than 100 times faster depending on n, m and p) by taking advantage of Matlab's vectorized capabilities. Also note that this function is sparse/structured D friendly because the only operation D is involved in is a matrix-vector product.

```
function mclogdet = DetAppxVectorized(D,p,m)
n = length(D);v = zeros(1,p);x = randn(n,p);
```

```
xTx = sum(x.*x);
c = x;
for j = 1:m
    c = D*c;
    v = v-n/j*sum(x.*c)./xTx;
end
mclogdet = mean(v);
```

And for a general SPD matrix generated by RBF interpolation with (8.12) (or another RBF) we can use all the adaptations we have talked about to come up with the final function. This Monte Carlo log determinant is used in the computation of the likelihood function in Section 8.5.3.

8.5 Inference based RBF parameterization involving eigenfunctions

The statistical methods discussed in Section 8.3.1 involved optimizing some function to determine an appropriate shape parameter: the kriging variance, the cross-validation error, or the likelihood function. Here the situation is the same, although we are concerned with the posterior distribution $p(\varepsilon, \sigma^2, \boldsymbol{y})$. Assuming the existence of some prior beliefs $p(\varepsilon, \sigma^2)$, even if they are completely uninformed, we can use Bayes' rule to write

$$\begin{aligned} p(\varepsilon, \sigma^2|\boldsymbol{y})p(y) &= p(\boldsymbol{y}|\varepsilon, \sigma^2)p(\varepsilon, \sigma^2) \\ p(\varepsilon, \sigma^2|\boldsymbol{y}) &\propto p(\boldsymbol{y}|\varepsilon, \sigma^2)p(\varepsilon, \sigma^2) \\ p(\varepsilon, \sigma^2|\boldsymbol{y}) &\propto L(\varepsilon, \sigma^2)p(\varepsilon, \sigma^2). \end{aligned}$$

We define $L(\varepsilon, \sigma^2)$ as the likelihood of the parameters given the data, and since we are assuming that the design points are not in our control, L is only a function of the parameters. Notice the presence of *proportional to* statements rather than equalities: because we are only interested in the shape of the posterior distribution we needn't carry around scaling or shifting terms.

We take advantage of our assumption that the data is generated by a Gaussian process. From earlier, we know the likelihood of the parameters given the data is

$$L(\varepsilon, \sigma^2) \propto \frac{\exp\left(-\frac{1}{2\sigma^2}\boldsymbol{y}^T\mathsf{K}^{-1}\boldsymbol{y}\right)}{(\sigma^2)^{n/2}\sqrt{\det(\mathsf{K})}}.$$

We may also make the assumption that σ^2 and ε are independent, which means that

$$p(\sigma^2, \varepsilon) = p(\sigma^2)p(\varepsilon).$$

If we make no prior assumptions about σ^2, that would yield the improper distribution

$$p(\sigma^2) = \begin{cases} 1, & \sigma^2 \geq 0 \\ 0, & \text{else} \end{cases}$$

which can be put in the posterior distribution

$$p(\varepsilon, \sigma^2|\boldsymbol{y}) \propto \frac{\exp\left(-\frac{1}{2\sigma^2}\boldsymbol{y}^T\mathsf{K}^{-1}\boldsymbol{y}\right)}{(\sigma^2)^{n/2}\sqrt{\det(\mathsf{K})}}p(\varepsilon), \qquad \sigma^2 \geq 0.$$

Because of the special structure of this distribution, σ^2 can be analytically integrated out (recall Section 8.2) to yield

$$\begin{aligned} P(\varepsilon) &= \int_0^\infty p(\varepsilon, \sigma^2|\boldsymbol{y})p(\varepsilon)d\sigma^2 \\ &= (\boldsymbol{y}^T\mathsf{K}^{-1}\boldsymbol{y})^{-n/2}\det(\mathsf{K})^{-1/2}p(\varepsilon), \end{aligned}$$

which is the distribution for the shape parameter given the assumptions we have made. Note that P can be interpreted as $p(\varepsilon|\boldsymbol{y})$, the profile posterior over all σ^2. The process of handling P is threatened by over/underflow, so instead we will evaluate its logarithm which is

$$\tilde{P}(\varepsilon) = -\frac{n}{2}\log(\boldsymbol{y}^T\mathsf{K}^{-1}\boldsymbol{y}) - \frac{1}{2}\log\det(\mathsf{K}) + \log p(\varepsilon). \tag{8.13}$$

Note the appearance of (8.5) in this equation, indicating that, as expected, the likelihood function plays a significant role in the posterior distribution. It, along with the K^{-1} term, will be handled using the stable basis where appropriate.

8.5.1 Prior knowledge

There is still a term $p(\varepsilon)$ in $\tilde{P}$ which in Bayesian inference is referred to as the prior distribution [80]. This can be used to incorporate any assumptions we make about ε based on physical characteristics of the underlying problem. For our purposes here, we are going to consider the use of 3 categories of priors and then choose an appropriate distribution. Possible options include

- Uninformative prior - $p(\varepsilon) = 1$
- Parametric prior - $p(\varepsilon) = \Gamma(\varepsilon; k, \theta)$
- Black Box prior - $p(\varepsilon) = \Psi(\varepsilon)$

If we have no knowledge of which shape parameter values are more appropriate, then the uninformative prior is the logical choice. As always, we need to be worried about the use of an improper prior distribution possibly yielding an improper posterior distribution (refer to [14]); here the form of the likelihood guarantees

a proper posterior. Even so, to assume that we know nothing about ε is to sell ourselves short: in [60] there are numerous examples with varying test functions, design points and kernels and the *optimal* shape parameter is rarely outside of $(0, 40)$. That is not to say it will always be that way, but rather that we should be able to make some judgments.

Given that we believe in some common range for ε, the question becomes: how do we incorporate the belief? Suppose our assumption is that the shape parameters will generally be in the range $(0, 40)$. One choice is to use a Gamma distribution, which allows for values outside that domain but has most of its mass centered around more likely values. See Figure 8.5 for an example. The distribution can be adjusted if we feel that the design points are on a smaller or larger scale such that different parameters would be appropriate.

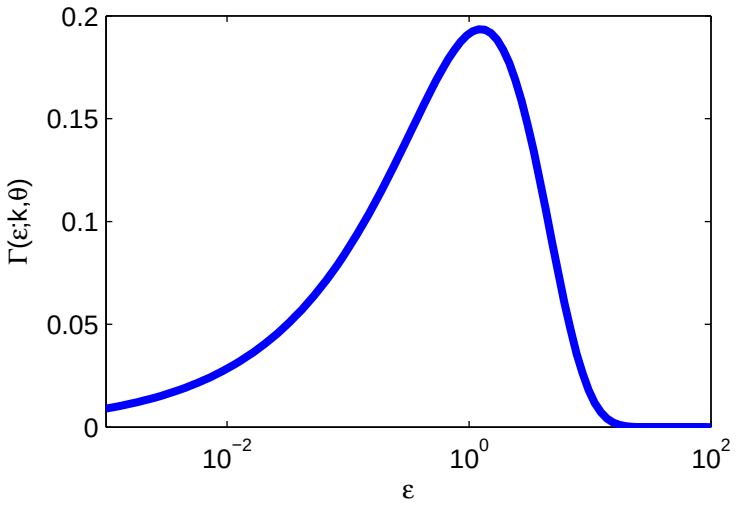


Figure 8.5: A sample Gamma prior distribution with mean 3.75 and mode 1.25, which helps enforce our belief that the optimal shape parameter will not be negative, nor greater than 40.

The third option listed earlier is the black box prior which loosely speaking involves using a complicated function to compute a $p(\varepsilon)$ value based on information which cannot be described using a parametric distribution. One example of this

might be to evaluate the power function (kriging variance) at various points in the domain and then use a weighted average as the $p(\varepsilon)$ value. The output of the function need not yield a proper prior, only values whose ratio indicates which choice of ε is preferred.

Because there are many possible choices for the black box prior (and they likely require extensive computation) we will focus our work on using

$$p(\varepsilon) = \Gamma(\varepsilon; k, \theta) = \frac{1}{\theta^k \Gamma(k)} \varepsilon^{k-1} \mathrm{e}^{-\varepsilon/\theta}$$

where $k = 1.5$ and $\theta = 2.5$ as depicted in Figure 8.5. This is appropriate for the demonstration in this work, but, in general, more application specific choices will need to be made for inference to be a valid method of parameter specification.

8.5.2 Using MCMC to simulate $p(\varepsilon|\boldsymbol{y})$

Our solution is to use Metropolis-Hastings (See [91],[120]) to generate a sequence of random variables $\{\varepsilon_i\}$ which follow the distribution $p(\varepsilon|\boldsymbol{y})$. The proposal distribution will be a normal distribution with mean at the current ε_i. The choice of variance ω^2 is vital because if our variance is too great then the random walk will take too few steps and if it is too small then the steps will be too short to fully explore the domain. This dilemma means that ω^2 will need to be problem dependent. Eventually, we hope to have some insight for a good value of ω, but for this thesis we will only choose a value of ω which, in retrospect, produces acceptable results.

Because random walk Metropolis-Hastings has a symmetric proposition, the

acceptance ratio for a step from ε_i to ε^* reduces to

$$\begin{aligned}\rho_i &= \frac{P(\varepsilon^*)N(\varepsilon_i;\varepsilon^*,\omega^2)}{P(\varepsilon_i)N(\varepsilon^*;\varepsilon_i,\omega^2)} \\ &= \frac{P(\varepsilon^*)}{P(\varepsilon_i)}.\end{aligned}$$

This means that at each step of the Markov chain we will need to evaluate $\tilde{P}$ once and draw one normal random variable. Then a random variable from the distribution $v_i \sim \text{Unif}(0,1)$ will be drawn, and if $v_i < \rho_i$, we take the step suggested and set $\varepsilon_{i+1} = \varepsilon^*$. Otherwise we set $\varepsilon_{i+1} = \varepsilon_i$.

8.5.3 A numerical example of inference based parameterization

At this point, we have defined all the necessary components to find the posterior distribution. Let's look at a specific 1D example of shape parameter selection by MCMC.

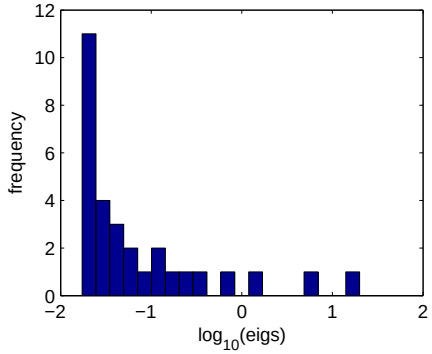
Suppose our design points/values are generated by the test function

$$f(x) = 1 + x$$

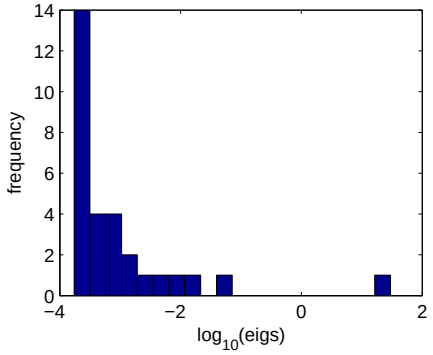
evaluated at 30 evenly spaced points x_k between $[0,1]$. If we choose to use truncated power functions as our kernel

$$K(x,z) = (1 - \varepsilon|x-z|)_+,$$

the spectrum of the resulting matrix is likely to have only a few eigenvalues greater than 1. Note that $(\mathsf{K})_{i,j} = K(x_i,x_j)$ is positive definite for x in one dimension [60], and sample spectra for $\varepsilon = 1$ and $\varepsilon = .01$ are shown in Figure 8.6. The condition numbers of those matrices are roughly 10^{-3} and 10^{-5} respectively.



(a) For $\varepsilon = 1$, three eigenvalues are greater than 1.



(b) For $\varepsilon = .01$, only one eigenvalue is greater than 1.

Figure 8.6: Only a few eigenvalues of K need to be handled with orthogonal draws for many ε values.

Using the techniques described earlier, we can evaluate $\tilde{P}$ from (8.13) for various ε (even though K may not be terribly ill-conditioned) and conduct an MCMC simulation to find $p(\varepsilon|\boldsymbol{y})$. The linear systems were solved with CG, and the determinant was computed statistically using the algorithm from Section 8.4.2. 1000 MCMC steps were conducted with 100 burn-in and the ω^2 variance in the random walk is chosen to be .1 to help the random walk sufficiently sample the domain. The resulting distribution can be found in Figure 8.7.

Using this distribution, we can make predictions of new y given an $x \notin x_1, ..., x_{30}$. Using the standard Bayesian practice for making predictions and (8.3), our prediction of y is

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} \hat{y}(\varepsilon_i),$$

where $\hat{y}(\varepsilon_i)$ is the best linear unbiased predictor for the i^{th} random variable drawn from $p(\varepsilon|\boldsymbol{y})$. The mean square error of our sample problem for $\hat{y}$ evaluated at 100 evenly spaced points is 6.6947×10^{-15}.

There is one obvious difficulty with this technique of predicting new (x, y)

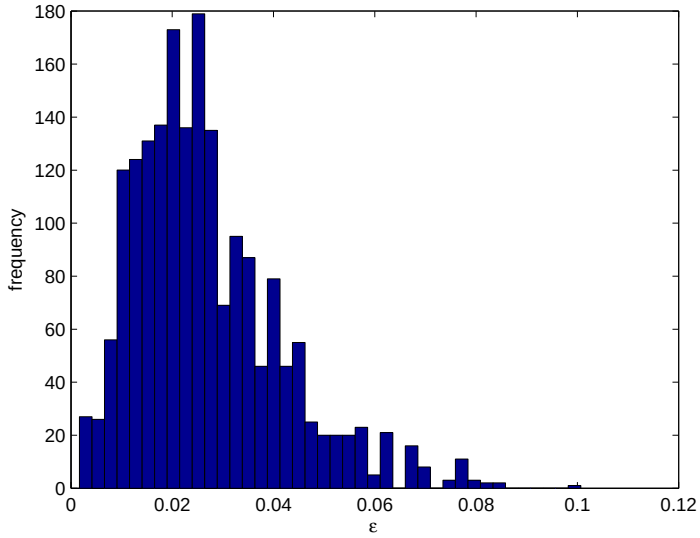


Figure 8.7: 900 random variables drawn from $p(\varepsilon|\boldsymbol{y})$ by the MCMC random walk.

values: each ε requires inverting the associated K. For the sample problem that means inverting 900 more (potentially ill-conditioned) matrices to accompany the 1000 inversions which took place in assembling $p(\varepsilon|\boldsymbol{y})$.

Rather than actually finding the best linear unbiased predictor, it is simpler to just approximate the mean of $p(\varepsilon|\boldsymbol{y})$ and use that value to make predictions. This is also closer to more standard techniques such as LOOCV and maximum likelihood estimation which produce a unique ε. When we compute the sample mean from our random walk, $\bar{\varepsilon} = .0267$ and the MS error of $\hat{y}(\bar{\varepsilon})$ for 100 points is 1.2789×10^{-14}. While this is not quite as good as the true estimator, the large reduction in work seems more valuable.

8.5.4 An inference example using the stable basis

We consider one additional example using the Hilbert-Schmidt SVD to approximate an associated posterior distribution. For this example, we will consider a simple scattered data interpolation problem involving $N = 30$ points at the Chebyshev nodes between $x \in [-3, 3]$. Our underlying function is simply $f(x) = \tanh(x)$, and we will compute the error at 100 evenly spaced points. Using GaussQR with $\alpha = 1$, we can produce Figure 8.8 which shows the error profile for various ε values.

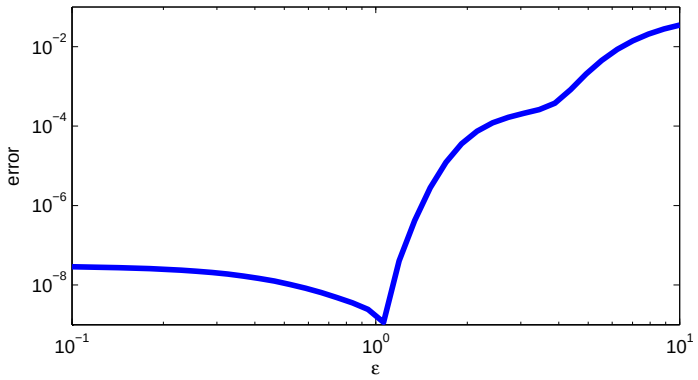


Figure 8.8: The function $f(x) = \tanh(x)$ can most accurately be approximated for $\varepsilon \approx 1$, when using GaussQR.

For this problem, the condition number of K is too large to compute (8.13) safely in the region of interest: even when $\varepsilon = 2$, $\kappa(\mathsf{K}) \approx 10^{14}$ which is too ill-conditioned to trust. We will instead use the stable basis to take draws from the posterior distribution $p(\varepsilon|\boldsymbol{y})$, specifically to compute the likelihood function as described in Section 8.3.2. The same prior distribution will be used as before, because we will assume no extra knowledge of the optimal ε.

We run a similar MCMC walk as before, with initial $\varepsilon_0 = 1$, random walk variance $\omega^2 = .002$, and 4000 total steps, 100 of which are discarded as burn-in. This resulted in an acceptance ratio of about 29%, which means that the simulation

was allowed to walk around without running rampant over low probability ε values. The simulated posterior distribution is found in Figure 8.9.

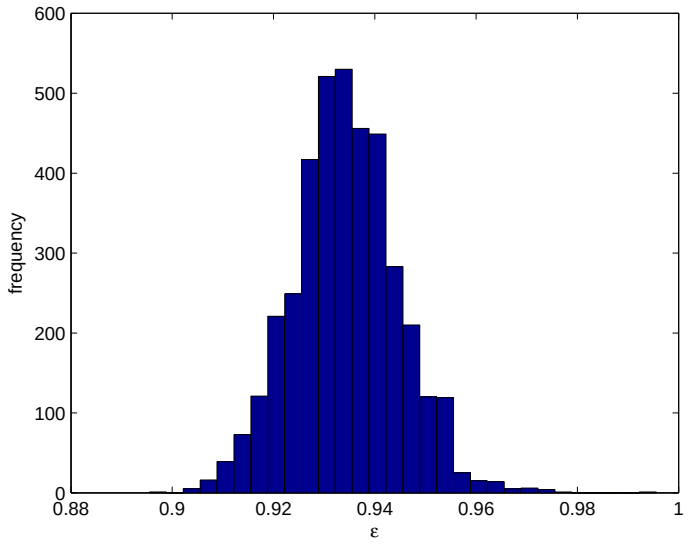


Figure 8.9: This posterior distribution suggests a good value for ε when compared to the error graph earlier. It does not find the optimal ε unfortunately, although it is close.

This posterior distribution is useful because it does suggest a region where the optimal ε may lie. This region is close to the true optimal ε, but referring back to Figure 8.8 reminds us that the optimal $\varepsilon > 1$, whereas the posterior distribution is firmly centered around $\varepsilon < 1$. These results are similar to the MLE results in Figure 8.3, where the optimal ε was near the maximum likelihood, but not actually there.

8.6 Summary

In this chapter we have rephrased the scattered data interpolation problem in the context of Gaussian processes. This allows us to consider statistical methods for

parameterizing our kernel-based approximation methods. We studied the effects of introducing the stable basis from Chapter 4 into this framework, and found that some statistical methods benefited, including cross-validation and maximum likelihood estimation. We have also described a technique for approximating the determinant of a positive definite matrix using statistics, which is useful in some circumstances when computing the likelihood function. Combining all these results, we have run stochastic simulations to approximate the posterior distribution $p(\varepsilon|\boldsymbol{y})$, which can be interpreted as the probability that a given ε produced the data $\boldsymbol{y}$. The results of these simulations are promising, but more work needs to be done to understand their usefulness. This work justifies inclusion in the multiphysics infrastructure by helping to improve our existing tools developed in Chapter 4, Chapter 6 and Chapter 7.

The experiments in this chapter only involved interpolation, but it seems reasonable to think that they could be extended for use on the BVP methods of Chapter 7. Although more difficult, because only part of the problem exists naturally within the Gaussian processes framework, it would also be possible to apply these ε predictions to the coupling methods of Chapter 6. A more significant step forward would be to incorporate the other free parameters α and M associated with the eigenfunction approach into a multivariate posterior distribution; this would help replace our current ad hoc strategy with a more formulaic approach. If optimal α, M and ε (or perhaps multiple ε_i if an anisotropic kernel is used) can be predicted using statistics, kernel- based methods will gain more acceptance in applications communities.

BIBLIOGRAPHY

[1] M. Abramowitz and I. A. Stegun. Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables. Applied mathematics series. Dover Publications, 1964.

[2] H. Adibi and J. Es'haghi. Numerical solution for biharmonic equation using multilevel radial basis functions and domain decomposition methods. Applied Mathematics and Computation, 186(1):246 – 255, 2007.

[3] R. Ahrem, A. Beckert, and H. Wendland. A new multivariate interpolation method for large-scale spatial coupling problems in aeroelasticity. In The Proceedings to Int. Forum on Aeroelasticity and Structural Dynamics, 2005.

[4] E. Alpaydin. Introduction to machine learning. Adaptive computation and machine learning. MIT Press, 2004.

[5] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM Journal on Matrix Analysis and Applications, 23(1):15–41, 2001.

[6] S. N. Atluri and T. Zhu. A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. Computational Mechanics, 22(2):117–127, August 1998.

[7] F. P. T. Baaijens. A fictitious domain/mortar element method for fluid-structure interaction. International Journal for Numerical Methods in Fluids, 35(7):743–761, 2001.

[8] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.2, Argonne National Laboratory, September 2011.

[9] M. T. Balhoff, S. G. Thomas, and M. F. Wheeler. Mortar coupling and upscaling of pore-scale models. Computers & Geosciences, 12:15–27, 2008.

[10] J. P. Bardhan, R. S. Eisenberg, and D. Gillespie. Discretization of the induced-charge boundary integral equation. Phys. Rev. E, 80:011906, Jul 2009.

[11] R. P. Barry and R. K. Pace. Monte Carlo estimates of the log determinant of large sparse matrices. Linear Algebra and its Applications, 289(13):41 – 54, 1999.

[12] R. K. Beatson, W. A. Light, and S. Billings. Fast solution of the radial basis function interpolation equations: Domain decomposition methods. SIAM J. Sci. Comput., 22:1717–1740, May 2000.

[13] T. Belytschko and S. P. Xiao. Coupling methods for continuum model with molecular model. International Journal for Multiscale Computational Engineering, 1(1), 2003.

[14] J. O. Berger, V. De Oliveira, and B. Sansó. Objective bayesian analysis of spatially correlated data. Journal of the American Statistical Association, 96(456):1361–1374, 2001.

[15] C. Bernardi, Y. Maday, and A. T. Patera. A new nonconforming approach to domain decomposition: the mortar element method. In H. Brezis and J.L. Lions, editors, Nonlinear PDEs and Their Applications. Longman, 1994.

[16] T. Betcke and L. N. Trefethen. Reviving the method of particular solutions. SIAM Review, 47(3):469 – 491, 2005.

[17] I. Bilionis and P. S. Koutsourelakis. Free energy computations by minimization of Kullback-Leibler divergence: An efficient adaptive biasing potential method for sparse representations. Journal of Computational Physics, 231(9):3849 – 3870, 2012.

[18] D. A. Bini, V. Mehrmann, V. Olshevsky, E. Tyrtsyhnikov, and M. Van Barel. Numerical Methods for Structured Matrices and Applications: The Georg Heinig Memorial Volume. Operator Theory: Advances and Applications. Birkhäuser, 2010.

[19] A. de Boer, A. H. van Zuijlen, and H. Bijl. Review of coupling methods for non-matching meshes. Comput. Methods Appl. Mech. Eng., 196(8):1515–1525, 2007.

[20] X. Bonnin, D. Coster, C. S. Pitcher, R. Schneider, D. Reiter, V. Rozhansky, S. Voskoboynikov, and H. Bürbaumer. Improved modelling of detachment and neutral-dominated regimes using the SOLPS/B2Eirene code. Journal of Nuclear Materials, 313-316(0):909 – 913, 2003.

[21] C. de Boor. On interpolation by radial polynomials. Advances in Computational Mathematics, 24:143–153, 2006.

[22] L. P. Bos and U. Maier. On the asymptotics of Fekete-type points for univariate radial basis interpolation. Journal of Approximation Theory, 119(2):252 – 270, 2002.

[23] J. P. Boyd. Error saturation in Gaussian radial basis functions on a finite interval. Journal of Computational and Applied Mathematics, 234(5):1435 – 1441, 2010.

[24] B. J. Braams. Radiative divertor modelling for ITER and TPX. Contributions to Plasma Physics, 36(2-3):276–281, 1996.

[25] S. I. Braginskii. Transport processes in a plasma. In M. A. Leontovich, editor, Reviews of plasma physics, volume 1, pages 205–311. Consultants Bureau, 1965.

[26] J. Broughton, F. Abraham, N. Bernstein, and E. Kaxiras. Concurrent coupling of length scales: Methodology and application. Physical Review B, 60(4):2391–2403, 1999.

[27] P. N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. SIAM Journal of Sci. Stat. Comput., 11(3):450–481, 1990.

[28] D. Brown (Chair). Applied Mathematics at the U.S. Department of Energy: Past, Present, and a View to the Future. Office of Science, U.S. Department of Energy, 2008.

[29] H. J. Bungartz, M. Mehl, and M. Schäfer. Fluid-Structure Interaction – Modelling, Simulation, Optimisation, Part II, volume 73 of Lecture Notes in Computational Science and Engineering. Springer, Berlin, Heidelberg, October 2010.

[30] W. Cai, M. de Koning, V. V. Bulatov, and S. Yip. Minimizing boundary reflections in coupled-domain simulations. Phys. Rev. Lett., 85:3213–3216, Oct 2000.

[31] X. C. Cai, D. E. Keyes, and L. Marcinkowski. Nonlinear additive Schwarz preconditioners and application in computational fluid dynamics. International Journal for Numerical Methods in Fluids, 40(12):1463–1470, 2002.

[32] D. Calvetti and L. Reichel. Fast inversion of Vandermonde-like matrices involving orthogonal polynomials. BIT Numerical Mathematics, 33:473–484, 1993. 10.1007/BF01990529.

[33] V. Carey, D. Estep, and S. Tavener. A posteriori analysis and adaptive error control for multiscale operator decomposition solution of elliptic systems I: triangular systems. SIAM J. Numer. Anal., 47(1):740–761, 2009.

[34] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM.

[35] J. R. Cary, J. Candy, R. H. Cohen, S. Krasheninnikov, D. C. McCune, D. J. Estep, J. Larson, A. D. Malony, P. H. Worley, J. A. Carlsson, A. H. Hakim, P. Hamill, S. Kruger, S. Muzsala, A. Pletzer, S. Shasharina, D. Wade-Stein, N. Wang, L. McInnes, T. Wildey, T. Casper, L. Diachin, T. Epperly, T. D. Rognlien, M. R. Fahey, J. A. Kuehn, A. Morris, S. Shende, E. Feibush, G. W. Hammett, K. Indireshkumar, C. Ludescher, L. Randerson, D. Stotler, A. Y. Pigarov, P. Bonoli, C. S. Chang, D. A. D'Ippolito, P. Colella, D. E. Keyes, R. Bramley, and J. R. Myra. Introducing FACETS, the framework application for core-edge transport simulations. Journal of Physics: Conference Series, 78(1):012086, 2007.

[36] J. R. Cary, A. Hakim, M. Miah, S. Kruger, A. Pletzer, S. Shasharina, S. Vadlamani, R. Cohen, T. Epperly, T. Rognlien, A. Pankin, R. Groebner, S. Balay, L. McInnes, and H. Zhang. FACETS - a framework for parallel coupling of fusion components. In Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP '10, pages 435–442, Washington, DC, USA, 2010. IEEE Computer Society.

[37] L. Chacón, D. A. Knoll, and J. M. Finn. An implicit, nonlinear reduced resistive MHD solver. Journal of Computational Physics, 178(1):15 – 36, 2002.

[38] C. S. Chang, S. Ku, and H. Weitzner. Numerical study of neoclassical plasma pedestal in a tokamak geometry. Physics of Plasmas, 11(5):2649–2667, 2004.

[39] C. S. Chen, H. A. Cho, and M. A. Golberg. Some comments on the ill-conditioning of the method of fundamental solutions. Engineering Analysis with Boundary Elements, 30(5):405–410, 2006.

[40] C. S. Chen, S. Lee, and C.-S. Huang. Derivation of particular solutions using Chebyshev polynomial based functions. Int. J. of Comp. Meth., 4(1):15–32, 2007.

[41] E. W. Cheney and W. A. Light. A Course in Approximation Theory. Graduate Studies in Mathematics. American Mathematical Society, 2000.

[42] P. Chidyagwai and B. Rivière. On the solution of the coupled Navier-Stokes and Darcy equations. Computer Methods in Applied Mechanics and Engineering, 198(47-48):3806 – 3820, 2009.

[43] S.-C. T. Choi. Iterative Methods for Singular Linear Equations and Least-Squares Problems. PhD thesis, ICME, Stanford University, CA, 2007.

[44] S.-C. T. Choi, C. C. Paige, and M. A. Saunders. MINRES-QLP: A Krylov subspace method for indefinite or singular symmetric systems. SIAM Journal on Scientific Computing, 33(4):1810–1836, 2011.

[45] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. SIAM Journal on Numerical Analysis, 20(1):pp. 187–209, 1983.

[46] T. M. Cover and J. A. Thomas. Elements of Information Theory. Wiley Series in Telecommunications and Signal Processing. John Wiley & Sons, 2006.

[47] J. Degroote, K. J. Bathe, and J. Vierendeels. Performance of a new partitioned procedure versus a monolithic procedure in fluid-structure interaction. Comput. Struct., 87:793–801, 2009.

[48] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact newton methods. SIAM Journal on Numerical Analysis, 19(2):pp. 400–408, 1982.

[49] C. J. Demeure. Fast QR factorization of Vandermonde matrices. Linear Algebra and its Applications, 122124(0):165 – 194, 1989.

[50] J. E. Dennis, Jr. and R. B. Schnabel. Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Classics in Applied Mathematics, 16). SIAM, 1996.

[51] M. R. Dorr, R. H. Cohen, P. Colella, M. A. Dorf, J. A. F. Hittinger, and D. F. Martin. Numerical simulation of phase space advection in gyrokinetic models

of fusion plasmas. In Proceedings of the 2010 Scientific Discovery through Advanced Computing (SciDAC) Conference, pages 42–52, July 2010, Oak Ridge, TN.

[52] T. A. Driscoll and B. Fornberg. Interpolation in the limit of increasingly flat radial basis functions. Computers & Mathematics with Applications, 43(3–5):413–422, 2002.

[53] D. Estep, V. Ginting, D. Ropp, J. N. Shadid, and S. Tavener. An a posteriori-a priori analysis of multiscale operator splitting. SIAM J. Numer. Anal., 46:1116–1146, March 2008.

[54] D. Estep, S. Tavener, and T. Wildey. A posteriori analysis and improved accuracy for an operator decomposition solution of a conjugate heat transfer problem. SIAM Journal on Numerical Analysis, 46(4):2068 – 2089, 2008.

[55] G. Fairweather and A. Karageorghis. The method of fundamental solutions for elliptic boundary value problems. Advances in Computational Mathematics, 9:69–95, 1998.

[56] R. D. Falgout. An introduction to algebraic multigrid. Computing in Science and Engineering, 8:24–33, November 2006.

[57] R. D. Falgout and U. M. Yang. HYPRE: A library of high performance preconditioners. In Proceedings of the International Conference on Computational Science-Part III, ICCS '02, pages 632–641, London, UK, 2002. Springer-Verlag.

[58] C. Farhat, M. Lesoinne, and P. Le Tallec. Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: Momentum and energy conservation, optimal discretization and application to aeroelasticity. Computer Methods in Applied Mechanics and Engineering, 157(1-2):95 – 114, 1998.

[59] G. E. Fasshauer. Solving partial differential equations by collocation with radial basis functions. In C. Rabut A. Le M'ehaut'e and L. L. Schumaker, editors, Surface Fitting and Multiresolution Methods, pages 131–138. University Press, 1997.

[60] G. E. Fasshauer. Meshfree Approximation Methods with MATLAB. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2007.

[61] G. E. Fasshauer, F. J. Hickernell, and H. Woźniakowski. On dimension-independent rates of convergence for function approximation with gaussian kernels. SIAM J. Numer. Anal., 50(1):247—271, 2012.

[62] G. E. Fasshauer, Fred J. Hickernell, and H. Woźniakowski. Average case approximation: convergence and tractability of Gaussian kernels. In H. Woźniakowski and L. Plaskota, editors, Monte Carlo and Quasi-Monte Carlo Methods 2010, volume 23 of Proceedings in Mathematics and Statistics, pages 309–324. Springer, Springer, 2012.

[63] G. E. Fasshauer and M. McCourt. Stable evaluation of gaussian RBF interpolants. SIAM J. Sci. Comput., 34(2):A737—A762, 2012.

[64] G. E. Fasshauer and L. L. Schumaker. Scattered data fitting on the sphere. In Proceedings of the International Conference on Mathematical Methods for Curves and Surfaces II Lillehammer, 1997, pages 117–166, Nashville, TN, USA, 1998. Vanderbilt University Press.

[65] Gregory Fasshauer and Jack Zhang. On choosing "optimal" shape parameters for RBF approximation. Numerical Algorithms, 45:345–368, 2007. 10.1007/s11075-007-9072-8.

[66] A. I. Fedoseyev, M. J. Friedman, and E. J. Kansa. Improved multiquadric method for elliptic partial differential equations via PDE collocation on the boundary. Computers & Mathematics with Applications, 43(35):439 – 455, 2002.

[67] N. Flyer and B. Fornberg. Radial basis functions: Developments and applications to planetary scale flows. Computers & Fluids, 46(1):23 – 32, 2011.

[68] N. Flyer, E. Lehto, S. Blaise, G. B. Wright, and A. St-Cyr. A guide to RBF-generated finite differences for nonlinear transport: Shallow water simulations on a sphere. Journal of Computational Physics, 231(11):4078 – 4095, 2012.

[69] B. Fornberg, T. A. Driscoll, G. Wright, and R. Charles. Observations on the behavior of radial basis function approximations near boundaries. Computers & Mathematics with Applications, 43(35):473 – 490, 2002.

[70] B. Fornberg, E. Larsson, and N. Flyer. Stable computations with Gaussian radial basis functions. SIAM Journal on Scientific Computing, 33(2):869–892, 2011.

[71] B. Fornberg and E. Lehto. Stabilization of RBF-generated finite difference methods for convective PDEs. Journal of Computational Physics, 230(6):2270 – 2285, 2011.

[72] B. Fornberg and C. Piret. A stable algorithm for flat radial basis functions on a sphere. SIAM J. Sci. Comput., 30:60–80, November 2007.

[73] B. Fornberg and C. Piret. On choosing a radial basis function and a shape parameter when solving a convective PDE on a sphere. Journal of Computational Physics, 227(5):2758 – 2780, 2008.

[74] B. Fornberg and G. Wright. Stable computation of multiquadric interpolants for all values of the shape parameter. Comput. Math. Appl., 48:853–867, September 2004.

[75] B. Fornberg and J. Zuev. The Runge phenomenon and spatially variable shape parameters in RBF interpolation. Comput. Math. Appl., 54:379–398, August 2007.

[76] Bengt Fornberg, Natasha Flyer, and Jennifer M. Russell. Comparisons between pseudospectral and radial basis function derivative approximations. IMA Journal of Numerical Analysis, 30(1):149–172, 2010.

[77] E. J. Fuselier and G. B. Wright. A High-Order Kernel Method for Diffusion and Reaction-Diffusion Equations on Surfaces. eprint arXiv:1206.0047, May 2012.

[78] D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandié. MOOSE: A parallel computational framework for coupled systems of nonlinear equations. Nuclear Engineering and Design, 239(10):1768 – 1778, 2009.

[79] W. Gautschi. The condition of Vandermonde-like matrices involving orthogonal polynomials. Linear Algebra and its Applications, 5253(0):293 – 300, 1983.

[80] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. Bayesian Data Analysis. Chapman & Hall/CRC, 2004.

[81] L. Gemignani. Fast QR factorization of Vandermonde-like matrices involving orthogonal polynomials. Technical Report TR-96-11, Universitá Di Pisa, Corso Italia 40,56125 Pisa, Italy, May 1996.

[82] M. A. Golberg and C. S. Chen. Discrete Projection Methods for Integral Equations. Computational Mechanics Publications, 1997.

[83] M. A. Golberg and C. S. Chen. The method of fundamental solutions for potential, Helmholtz and diffusion problems. Computational engineering. Computational Mechanics Publications, WIT Press, 1998.

[84] G. H. Golub and C. F. Van Loan. Matrix computations (3rd ed.). Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[85] W. Gropp, D. Keyes, L. C. McInnes, and M. D. Tidriri. Globalized Newton-Krylov-Schwarz algorithms and software for parallel implicit CFD. Int. J. High Perform. Comput. Appl., 14:102–136, May 2000.

[86] Park H., Knoll D. A., Gaston D. R., and Martineau R. C. Tightly coupled multiphysics algorithms for pebble bed reactors. Nuclear Science and Engineering, 166(2):118–133, 2010.

[87] W. S. Hall. The Boundary Element Method. Solid Mechanics and its Applications. Kluwer Academic Publishers, 1994.

[88] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numerische Mathematik, 2:84–90, 1960. 10.1007/BF01386213.

[89] G. Hammond. Innovative methods for solving multicomponent biogeochemical groundwater transport on supercomputers. PhD thesis, University of Illinois at Urbana-Champaign, 2003.

[90] G. E. Hammond, A. J. Valocchi, and P. C. Lichtner. Application of Jacobian-free Newton-Krylov with physics-based preconditioning to biogeochemical transport. Advances in Water Resources, 28(4):359–376, 2005.

[91] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. Biometrika, 57(1):97–109, 1970.

[92] N. J. Higham and F. Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. SIAM J. Matrix Anal. Appl., 21:1185–1201, March 2000.

[93] Y. C. Hon. A quasi-radial basis functions method for American options

pricing. Computers & Mathematics with Applications, 43(35):513 – 524, 2002.

[94] Y. C. Hon and R. Schaback. On unsymmetric collocation by radial basis functions. Applied Mathematics and Computation, 119(23):177 – 186, 2001.

[95] T. M. Huang, V. Kecman, and I. Kopriva. Kernel Based Algorithms for Mining Huge Data Sets: Supervised, Semi-supervised, and Unsupervised Learning. Studies in Computational Intelligence. Springer, 2006.

[96] M. S. Ingber and N. Phan-Thien. A boundary element approach for parabolic differential equations using a class of particular solutions. Applied Mathematical Modelling, 16(3):124 – 132, 1992.

[97] A. Iserles. A first course in the numerical analysis of differential equations. Cambridge texts in applied mathematics. Cambridge University Press, 2009.

[98] R. K. Jaiman, X. Jiao, P. H. Geubelle, and E. Loth. Conservative load transfer along curved fluid-solid interface with non-matching meshes. J. Comp. Phys., 218(1):372–397, 2006.

[99] S. C. Jardin, G. Bateman, G. W. Hammett, and L. P. Ku. On 1D diffusion problems with a gradient-dependent diffusion coefficient. Journal of Computational Physics, 227(20):8769 – 8775, 2008.

[100] B. Jumarhon, S. Amini, and K. Chen. The Hermite collocation method using radial basis functions. Engineering Analysis with Boundary Elements, 24(78):607 – 611, 2000.

[101] S. Y. Kadioglu and D. A. Knoll. A fully second order implicit/explicit time integration technique for hydrodynamics plus nonlinear heat conduction problems. Journal of Computational Physics, 229(9):3237 – 3249, 2010.

[102] T. Kailath and A. H. Sayed. Fast Reliable Algorithms for Matrices With Structure. Advances in Design and Control. Society for Industrial and Applied Mathematics, 1999.

[103] E. J. Kansa. Multiquadrics–A scattered data approximation scheme with applications to computational fluid-dynamics–II solutions to parabolic, hyperbolic and elliptic partial differential equations. Computers & Mathematics with Applications, 19(89):147 – 161, 1990.

[104] C. T. Kelley. Iterative methods for linear and nonlinear equations, volume 16 of Frontiers in applied mathematics. Society for Industrial and Applied Mathematics, 1995.

[105] C. T. Kelley. Solving nonlinear equations with Newton's method. Fundamentals of algorithms. Society for Industrial and Applied Mathematics, 2003.

[106] D. E. Keyes, L. C. McInnes, C. Woodward, W. D. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawlowski, A. Peters, D. Reynolds, B. Riviere, U. Rüde, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, and B. Wohlmuth. Multiphysics simulations: Challenges and opportunities. Technical Report ANL/MCS-TM-321, Argonne National Laboratory, Dec 2011.

[107] D. R. Kincaid and E. W. Cheney. Numerical analysis: mathematics of scientific computing. Pure and applied undergraduate texts. American Mathematical Society, 2002.

[108] T. Klöppel, A. Popp, U. Küttler, and W. A. Wall. Fluid-structure interaction for non-conforming interfaces based on a dual mortar formulation. Computer Methods in Applied Mechanics and Engineering, 2011. accepted.

[109] D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. J. Comput. Phys., 193:357–397, January 2004.

[110] R. Krasny and L. Wang. Fast evaluation of multiquadric RBF sums by a Cartesian treecode. SIAM J. Scientific Computing, 33(5):2341–2355, 2011.

[111] R. Krupiński and J. Purczyński. Approximated fast estimator for the shape parameter of generalized Gaussian distribution. Signal Processing, 86(2):205 – 211, 2006.

[112] E. Larsson and B. Fornberg. A numerical study of some radial basis function based solution methods for elliptic PDEs. Computers & Mathematics with Applications, 46(5–6):891–902, 2003.

[113] E. Larsson and B. Fornberg. Theoretical and computational aspects of mul-

tivariate interpolation with increasingly flat radial basis functions. Comput. Math. Appl., 49:103–130, January 2005.

[114] E. Larsson and A. Heryudono. A partition of unity radial basis function collocation method for partial differential equations. 2013. in preparation.

[115] Y. J. Lee, G. J. Yoon, and J. Yoon. Convergence of increasingly flat radial basis interpolants to polynomial interpolants. SIAM Journal on Mathematical Analysis, 39(2):537–553, 2007.

[116] R. J. LeVeque. Finite volume methods for hyperbolic problems. Cambridge texts in applied mathematics. Cambridge University Press, 2002.

[117] J. Li and Y. C. Hon. Domain decomposition for radial basis meshless methods. Numerical Methods for Partial Differential Equations, 20(3):450–462, 2004.

[118] X. Li and C. Pozrikidis. The effect of surfactants on drop deformation and on the rheology of dilute emulsions in Stokes flow. Journal of Fluid Mechanics, 341:165–194, 1997.

[119] L. Ling and M. R. Trummer. Adaptive multiquadric collocation for boundary layer problems. Journal of Computational and Applied Mathematics, 188(2):265 – 282, 2006.

[120] J. S. Liu. Monte Carlo Strategies in Scientific Computing. Springer Publishing Company, Incorporated, 2008.

[121] J. C. Mairhuber. On Haar's theorem concerning Chebychev approximation problems having unique solutions. Proceedings of the American Mathematical Society, 7(4):609–615, 1956.

[122] M. McCourt. A fast least squares solver for stable Gaussian computations. In Proceedings of the Eighth International Conference on Scientific Computing and Applications, Las Vegas, Nevada, April 2012. submitted.

[123] M. McCourt. Stable Gaussians for boundary value problems. Advanced in Applied Mathematics and Mechanics, 2012. accepted.

[124] M. McCourt, T. D. Rognlien, L. C. McInnes, and H. Zhang. Improving parallel scalability for edge plasma transport simulations with neutral gas species. Computational Science & Discovery, 5(1):014012, 2012.

[125] G. McKee, K. Burrell, R. Fonck, G. Jackson, M. Murakami, G. Staebler, D. Thomas, and P. West. Impurity-induced suppression of core turbulence and transport in the DIII-D tokamak. Phys. Rev. Lett., 84:1922–1925, Feb 2000.

[126] P. D. McNelis. Neural Networks in Finance: Gaining Predictive Edge in the Market. Academic Press Advanced Finance Series. Elsevier Science, 2004.

[127] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character, 209(441–458):415–446, 1909.

[128] A. Miele and R. R. Iyer. General technique for solving nonlinear, two-point boundary-value problems via the method of particular solutions. Journal of Optimization Theory and Applications, 5:382–399, 1970. 10.1007/BF00928674.

[129] V. A. Mousseau and D. A. Knoll. New physics-based preconditioning of implicit methods for non-equilibrium radiation diffusion. Journal of Computational Physics, 190(1):42 – 51, 2003.

[130] B. Nagy, J. L. Loeppky, and W. J. Welch. Fast bayesian inference for gaussian process models. Technical Report 230, University of British Columbia, Department of Statistics, June 2007.

[131] A. Neumaier. Solving ill-conditioned and singular linear systems: A tutorial on regularization. SIAM Review, 40(3):636–666, 1998.

[132] P. W. Partridge, C. A. Brebbia, and L. C. Wrobel. The dual reciprocity boundary element method. International series on computational engineering. Computational Mechanics Publications, 1992.

[133] S. V. Patankar. Numerical heat transfer and fluid flow. Series in computational methods in mechanics and thermal sciences. Taylor & Francis, 1980.

[134] M. Pazouki and R. Schaback. Bases for kernel-based spaces. J. Comput. Appl. Math., 236(4):575–588, September 2011.

[135] M. Pernice and H. F. Walker. NITSOL: A Newton iterative solver for nonlinear systems. SIAM J. Sci. Comput., 19:302–318, January 1998.

[136] V. Pták. Lyapunov, Bézout, and Hankel. Linear Algebra and its Applications, 58:363 – 390, 1984.

[137] C. E. Rasmussen and C. K. I. Williams. Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press, 2005.

[138] D. Reiter, M. Baelmans, and P. Börner. The EIRENE and B2-EIRENE codes. Fusion Science and Technology, 47(2):172–186, 2005.

[139] J. D. Riley. Solving systems of linear equations with a positive definite, symmetric, but possibly ill-conditioned matrix. Mathematical Tables and Other Aids to Computation, 9(51):pp. 96–101, 1955.

[140] S. Rippa. An algorithm for selecting a good value for the parameter c in radial basis function interpolation. Advances in Computational Mathematics, 11:193–210, 1999. 10.1023/A:1018975909870.

[141] B. Rodhe. A discontinuous Galerkin method with local radial basis function interpolation. Uptec report f 07 066, School of Engineering, Uppsala Univ., Uppsala, Sweden, 2007, 2007.

[142] T. D. Rognlien, J. L. Milovich, M. E. Rensink, and G. D. Porter. A fully implicit, time dependent 2-D fluid code for modeling tokamak edge plasmas. Journal of Nuclear Materials, 196-198(0):347 – 351, 1992.

[143] T. D. Rognlien and M. E. Rensink. Edge-plasma models and characteristics for magnetic fusion energy devices. Fusion Engineering and Design, 60(4):497 – 514, 2002.

[144] T. D. Rognlien, X. Q. Xu, and A. C. Hindmarsh. Application of parallel implicit methods to edge-plasma numerical simulations. J. Comput. Phys., 175:249–268, January 2002.

[145] R. Ross. Formulas to describe the bias and standard deviation of the ML-estimated Weibull shape parameter. Dielectrics and Electrical Insulation, IEEE Transactions on, 1(2):247 –253, apr 1994.

[146] R. E. Rudd. The atomic limit of finite element modeling in mems: Coupling of length scales. Analog Integr. Circuits Signal Process., 29:17–26, October 2001.

[147] Y. Saad. ILUT: A dual threshold incomplete LU factorization. Numerical Linear Algebra with Applications, 1(4):387–402, 1994.

[148] Y. Saad. Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[149] S. A. Sarra. A numerical study of the accuracy and stability of symmetric and asymmetric RBF collocation methods for hyperbolic pdes. Numerical Methods for Partial Differential Equations, 24(2):670–686, 2008.

[150] S. A. Sarra and D. Sturgill. A random variable shape parameter strategy for radial basis function approximation methods. Engineering Analysis with Boundary Elements, 33(11):1239 – 1245, 2009.

[151] R. Schaback. Error estimates and condition numbers for radial basis function interpolation. Advances in Computational Mathematics, 3:251–264, 1995. 10.1007/BF02432002.

[152] R. Schaback. Convergence of unsymmetric kernel-based meshless collocation methods. SIAM J. Numer. Anal., 45:333–351, January 2007.

[153] R. Schaback. Limit problems for interpolation by analytic radial basis functions. J. Comput. Appl. Math., 212:127–149, February 2008.

[154] R. Schaback. Unsymmetric meshless methods for operator equations. Numerische Mathematik, 114:629–651, 2010. 10.1007/s00211-009-0265-z.

[155] R. Schaback. Approximationsverfahren II. Institut für Numerische und Angewandte Mathematik, Universität Göttingen, 2011.

[156] M. Schäfer, D. C Sternel, G Becker, and P. Pironkov. Efficient numerical simualtion and optimization of fluid-structure interaction. In H.-J. Bungartz, M. Mehl, and M. Schäfer, editors, Fluid-Structure Interaction II, number 73 in Lecture Notes in Computational Science and Engineering, pages 131–158, Berlin, Heidelberg, 2010. Springer.

[157] E. Schmidt. Über die Auflösung linearer Gleichungen mit unendlich vielen Unbekannten. Palermo Rend., 25:53–77, 1908.

[158] K. Sharifi and A. Leon-Garcia. Estimation of shape parameter for generalized Gaussian distributions in subband decompositions of video. IEEE Trans. Cir. and Sys. for Video Technol., 5(1):52–56, February 1995.

[159] A. Shokri and M. Dehghan. A Not-a-Knot meshless method using radial basis functions and predictorcorrector scheme to the numerical solution of improved Boussinesq equation. Computer Physics Communications, 181(12):1990 – 2000, 2010.

[160] H. Simon, T. Zacharia, and R. Stevens (Co-Chairs). Modeling and Simulation at the Exascale for Energy and the Environment. Office of Science, U.S. Department of Energy, 2007.

[161] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. Domain Decomposition: Parallel multilevel methods for elliptic partial differential equations. Cambridge University Press, New York, NY, USA, 1996.

[162] P. C. Stangeby. The plasma boundary of magnetic fusion devices. Series On Plasma Physics. Institute of Physics Pub., 2000.

[163] M. L. Stein. Interpolation of Spatial Data: Some Theory for Kriging. Springer Series in Statistics. Springer, 1999.

[164] D. P. Stotler, C. F. F. Karney, M. E. Rensink, and T. D. Rognlien. Coupling of parallelized DEGAS 2 and UEDGE codes. Contributions to Plasma Physics, 40(3-4):221–226, 2000.

[165] G. Szegő. Orthogonal polynomials. Colloquium Publications - American Mathematical Society. American Mathematical Society, 1939.

[166] C. J. Trahan and R. E. Wyatt. Radial basis function interpolation in the quantum trajectory method: optimization of the multi-quadric shape parameter. Journal of Computational Physics, 185(1):27 – 49, 2003.

[167] J. F. Traub, G. W. Wasilkowski, and H. Woźniakowski. Information-Based Complexity. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[168] L. N. Trefethen. Spectral Methods in Matlab. Software, Environments, Tools. Society for Industrial and Applied Mathematics, 2000.

[169] L. N. Trefethen and D. Bau III. Numerical linear algebra. SIAM, 1997.

[170] E. H. van Brummelen. Partitioned iterative solution methods for fluid-structure interaction. Int. J. Numer. Meth. Fluids, 65:3–27, 2010.

[171] C. F. Van Loan. The ubiquitous Kronecker product. Journal of Computational and Applied Mathematics, 123(12):85 – 100, 2000.

[172] W. A. Wall, D. P. Mok, and E. Ramm. Accelerated iterative substructuring schemes for instationary fluid-structure interaction. Computational Fluid and Solid Mechanics, pages 1325–1328, 2001.

[173] J. G. Wang and G. R. Liu. On the optimal shape parameters of radial basis functions used for 2-D meshless methods. Computer Methods in Applied Mechanics and Engineering, 191(2324):2611 – 2630, 2002.

[174] X. S. Wang. An iterative matrix-free method in implicit immersed boundary/continuum methods. Computers & Structures, 85(11-14):739 – 748, 2007.

[175] W. Washington (Chair). Scientific Grand Challenges: Challenges in Climate Change Science and the Role of Computing at the Extreme Scale. Office of Science, U.S. Department of Energy, 2008.

[176] H. Wendland. Scattered Data Approximation. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005.

[177] H. Wendland. Spatial coupling in aeroelasticity by meshless kernel-based methods. In E. aan Zee, editor, ECCOMAS CFD, The Netherlands, 2006.

[178] H. Wendland. Hybrid methods for fluid-structure-interaction problems in aeroelasticity. In Michael Griebel and Marc Alexander Schweitzer, editors, Meshfree Methods for Partial Differential Equations IV, volume 65 of Lecture Notes in Computational Science and Engineering, pages 335–358. Springer Berlin Heidelberg, 2008.

[179] L. B. Wigton, N. J. Yu, and D. P. Young. GMRES acceleration of computational fluid dynamics codes. In AIAA 7th Computational Fluid Dynamics Conference, 1985.

[180] T.-T. Wong, W.-S. Luk, and P.-A. Heng. Sampling with Hammersley and Halton points. Journal of Graphics Tools, 2(2):9–24, 1997.

[181] G. B. Wright and B. Fornberg. Scattered node compact finite difference-type formulas generated from radial basis functions. Journal of Computational Physics, 212(1):99 – 123, 2006.

[182] Y. Xu. A matrix free Newton/Krylov method for coupling complex multi-physics subsystems. PhD thesis, Purdue University, 2004.

[183] A. Yeckel, L. Lun, and J. J. Derby. An approximate block Newton method for coupled iterations of nonlinear solvers: Theory and conjugate heat transfer applications. Journal of Computational Physics, 228(23):8566 – 8588, 2009.

[184] H. Zhu, C. K. I. Williams, R. J. Rohwer, and M. Morciniec. Gaussian regression and optimal finite dimensional linear models. In C. M. Bishop, editor, Neural Networks and Machine Learning. Springer-Verlag, Berlin, 1998.

[185] B. Zwicknagl. Power series kernels. Constructive Approximation, 29:61–84, 2009. 10.1007/s00365-008-9012-4.