Part I

Preliminaries

Chapter 1

RBF Methods for PDEs

The process of solving PDEs using RBFs dates back to 1990 [**?**, **?**]. However, the This chapter starts with a description of the general approximation problem and provide background on RBF scattered data interpolation that will be required for the remainder of the chapter. We categorize existing methods for solving PDEs with RBFs as either global or local. Global methods use collocation and invert a single large linear system to find the interpolant that satisfies the differential equations at RBF centers. Local methods limit the influence of basis functions and seek an interpolant at each RBF center defined in terms of neighboring basis functions (local collocation) or nodal values (RBF-FD).

We present three collocation methods: Kansa's method, Fasshauer's method and Direct collocation, and discuss extensions that lead to local interpolation matrices instead of a single global interpolation matrix. Finally, we discuss RBF-FD, the most recent method for solving PDEs.

1.1 PDE as an Interpolation Problem

Following [**?**], consider a PDE expressed in terms of a (linear) differential operator, $\mathcal{L}$:

$$\mathcal{L}u = f \quad \text{on } \Omega \tag{1.1}$$

$$u = g \quad \text{on } \Gamma \tag{1.2}$$

where Ω is the interior of the physical domain, Γ is the boundary of Ω and f, g are known explicitly. In the case of a non-linear differential operator, a Newton's iteration, or some other method, can be used to linearize the problem (see e.g., [**?**]); of course, this increases the complexity of a single time step. Then, the unknown solution, u, which produces the observations on the right hand side can be approximated by an interpolant function u_Φ expressed as a linear combination of radial basis functions, $\{\Phi_j(x) = \Phi(||x - x_j||)\}_{j=1}^N$, and

polynomial functions$\{P_l(x)\}_{l=1}^{M}$:

$$u_\Phi(x) = \sum_{j=1}^{N} c_j \Phi_j(x) + \sum_{l=1}^{M} d_l P_l(x), \qquad P_l(x) \in \Pi_p^D \tag{1.3}$$

where $\Phi_j(x) = ||x - x_j||$ ($||\cdot||$ is standard Euclidean distance). The second sum represents a linear combination of polynomials that enforces zero approximation error when $u(x)$ is a polynomial of degree than or equal to p. The variable D is the problem dimension (i.e., $u_\Phi(x) \in \mathbb{R}^D$). To eliminate degrees of freedom for well-posedness, p should be greater than or equal to the order of the chosen RBF (see Table **??**) [**?**]. Note that Equation 1.3 is evaluated at $x = \{x_i\}_{i=1}^{n}$ known data points through which the interpolant is required to pass with zero residual. We refer to the x_i's (data points) points as *collocation points*, usually taken as the RBF centers, although this is not a requirement.

To clarify the role of the polynomial part in Equation 1.3, it is necessary to put aside the PDE for the moment and consider only the problem of *scattered data interpolation* with Radial Basis Functions. Borrowing notation from [**?**, **?**], we seek an interpolant, $\mathcal{P}_f$ of the form

$$\begin{aligned} \mathcal{P}_f &= \sum_{j=1}^{N} c_j \Phi_j(x) = f(x) \\ &= \sum_{j=1}^{N} \Phi_j(x) c_j = f(x) \\ \vec{\mathcal{P}_f} &= \Phi^T(x) c = f(x) \end{aligned}$$

where $\mathcal{P}_f$ is expressed as a scalar product between the unknown coeffient weights c_j and the radial basis functions $\Phi_j(x)$.

In many cases it is desirable to exactly reproduce functions of degree less than or equal to some order m. For RBF scattered data interpolation in one dimension, this can be achieved by adding a polynomial of order m with $M = \binom{m+1}{1}$ terms (e.g., $x^0, x^1, \cdots, x^m$). In 2D, the terms would be: $1, x, y, xy, x^2y, xy^2, \cdots, x^m y^{m-1}, x^{m-1}y^m, x^m y^m$. In $\mathbb{R}^D$, $M = \binom{m+D}{D}$ [**?**], giving

$$f(x) = \sum_{j=1}^{N} c_j \Phi_j(x) + \sum_{l=1}^{M} d_l P_l(x), \qquad P_l(x) \in \Pi_m^D \tag{1.4}$$

$$f = \Phi^T c + P^T d \tag{1.5}$$

where the second summation (referred to as *interpolation conditions* [**?**]) ensures the minimum degree of the interpolant. Notice, however, that the interpolation conditions add M new degrees of freedom, so we must provide M *moment conditions* or constraints:

$$\sum_{j=1}^{N} c_j P_l(x_j) = 0, \qquad l = 1, ..., M$$

or

$$\mathbb{P}^T c = 0$$

to guarantee that the moments of the interpolant vanish for the coefficients obtained, where the matrix

$$\mathbb{P}^T = \begin{pmatrix} P(x_1) & P(x_2) & \cdots & P(x_N) \end{pmatrix}$$

and

$$P(x_j) = \begin{pmatrix} P_1(x_j) & P_2(x_j) & \cdots & P_M(x_j) \end{pmatrix}^T.$$

It is now possible again to write the interpolation as a linear system using Equations 1.5 and 1.1:

$$\vec{\mathcal{P}_f} = \begin{bmatrix} \Phi & \mathbb{P} \\ \mathbb{P}^T & 0 \end{bmatrix} \begin{pmatrix} \vec{c} \\ \vec{d} \end{pmatrix} = \begin{pmatrix} \vec{f} \\ 0 \end{pmatrix} \tag{1.6}$$

This system then produces an interpolant capable of exactly approximating data from polynomials of degree less than or equal to m [**?**].

Returning to the PDE, we postpone the issue of achieving polynomial precision in approximation. However, it has been shown (see [**?**, **?**]) that some choices of RBFs (e.g. multiquadrics and thin-plate splines [**?**]) are not positive definite and therefore there is no guarantee that the approximation is well-posed. A sufficient condition for well-posedness is that the matrix is *conditionally positive definite*. In [**?**], Fasshauer demonstrates that conditional positive definiteness of order m is guaranteed when Equation 1.3 includes the full polynomial part (i.e., interpolation conditions) to match the order of the RBF.

Now, since $u_\Phi(x)$ from Equation 1.3 cannot (in general) satisfy the PDE everywhere, we enforce the PDE at a set of collocation points, which are distributed over both the interior and the boundary. Again, these points do not necessarily coincide with the RBF centers, but it is convenient for this to be true in practice.

1.2 Reconstructing Solutions for PDEs

In the next few sections, we will generate collocation equations based on this general form:

$$\mathcal{L}u_\Phi(x) = f(x) \quad \text{on } \Omega \tag{1.7}$$
$$\mathcal{B}u_\Phi(x) = g(x) \quad \text{on } \Gamma \tag{1.8}$$

where the methods presented below will apply the differential operators, $\mathcal{L}$ and $\mathcal{B}$, to different choices of u_Φ and different sets of collocation points. In many applications $\mathcal{L}$ is chosen as a differential operator (e.g., ∇, ∇^2) and $\mathcal{B} = I$ (i.e. identity operator for Dirichlet boundary conditions) for PDEs. There are also applications where $\mathcal{L}$ is a convolution operator (see e.g., [**?**, **?**]) capable of smoothing/de-noising a surface reconstructed from point clouds.

For all the methods presented here a linear system is generated

$$A \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} c \\ d \end{pmatrix} = A^{-1} \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{1.9}$$

where matrix A depends on the choice of collocation method. Once the linear system is solved, the value, $u(x_i)$, at an unknown point, x_i, can be reconstructed using the coefficients with the following inner product:

$$u(x_i) = \begin{bmatrix} \Phi^T(x_i) & P^T(x_i) \end{bmatrix} \begin{pmatrix} c \\ d \end{pmatrix} \tag{1.10}$$

Likewise, to find differential quantities we have:

$$\mathcal{L}u(x_i) = \begin{bmatrix} \mathcal{L}(\Phi^T)(x_i) & \mathcal{L}(P^T)(x_i) \end{bmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = A_{\mathcal{L}} \begin{pmatrix} c \\ d \end{pmatrix} \tag{1.11}$$

Here we substitute $A_{\mathcal{L}}$ for the right hand side row vector. Also, Equation 1.9 can be substituted into Equation 1.11 to get

$$\mathcal{L}u(x_i) = A_{\mathcal{L}} A^{-1} \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{1.12}$$

where the vector-matrix inner product $(A_{\mathcal{L}} A^{-1})$ is a row-vector. Since the coefficient vectors c and d are the same for all x_i, we can group the evaluation of $\mathcal{L}u(x_i)$ for $i = 1, ..., n$ as a matrix-vector multiplication where the matrix rows correspond to $(A_{\mathcal{L}} A^{-1})$ for each x_i.

1.3 Kansa's Method

The first method of collocation, *Kansa's method* [**?**, **?**], collocates the solution through known values on the boundary, while constraining the interpolant to satisfy the PDE operator on the interior. This is equivalent to choosing u_Φ according to Equation 1.3. The resulting system is given by [**?**]; assuming that $\mathcal{L}$ is a linear operator,

$$\mathcal{L}u_\Phi(x_i) = \sum_{j=1}^{N} c_j \mathcal{L}\Phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{L}P_l(x_i) = f(x_i) \qquad i = 1, ..., n_I \tag{1.13}$$

$$\mathcal{B}u_\Phi(x_i) = \sum_{j=1}^{N} c_j \mathcal{B}\Phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{B}P_l(x_i) = g(x_i) \qquad i = n_I + 1, \cdots, n \tag{1.14}$$

$$\sum_{j=1}^{N} c_j P_l(x_j) = 0 \qquad l = 1, \cdots, M \tag{1.15}$$

where n_I are the number of interior collocation points, with the number of boundary collocation points equal to $n - n_I$. First, observe that the differential operators are applied directly to the RBFs inside summations, rather than first solving the scattered data interpolation problem and then applying the operator to the interpolant. Second, since the basis functions are known analytically, it is possible (although sometimes painful) to derive $\mathcal{L}\Phi$ (refer to [?] for RBF derivative tables); the same is true for the polynomials P_l.

We can now reformulate Kansa's method as the linear system:

$$\begin{bmatrix} \Phi_{\mathcal{L}} & P_{\mathcal{L}} \\ \Phi_{\mathcal{B}} & P_{\mathcal{B}} \\ P^T & 0 \end{bmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} f \\ g \\ 0 \end{pmatrix} \tag{1.16}$$

where $\Phi_{\mathcal{L}} = \mathcal{L}\Phi$, $P_{\mathcal{L}} = \mathcal{L}P$ are the interior components (Equation 1.13), $\Phi_{\mathcal{B}}$ and $P_{\mathcal{B}}$ are the boundary components (Equation 1.14), and $P^T = \begin{bmatrix} P_{\mathcal{L}}^T & P_{\mathcal{B}}^T \end{bmatrix}$ are moment constraints for both interior and boundary polynomial parts (Equation 1.15). From Equation 1.16 it should be clear why Kansa's method is also known as the *Unsymmetric* collocation method.

Recall from Chapter **??** that the matrix in Equation 1.16 has no guarantee of non-singularity [?]; however, singularities are rare in practice [?].

1.4 Fasshauer's Method

Fasshauer's method [?] addresses the problem of singularity in Kansa's method by assuming the interpolation to be Hermite. That is, it requires higher differentiability of the basis functions (they must be at least C^k-continuous if $\mathcal{L}$ is of order k). Leveraging this assumption, Fasshauer's method chooses:

$$u_\Phi(x_i) = \sum_{j=1}^{N_I} c_j \mathcal{L}\Phi_j(x_i) + \sum_{j=N_I+1}^{N} c_j \mathcal{B}\Phi_j(x_i) + \sum_{l=1}^{M} d_l P_l(x_i) \tag{1.17}$$

as the interpolant passing through collocation points. Note N_I is used here to specify the number of RBF centers in the interior of Ω. Here the interpolant is similar to Equation 1.3, but a change of basis functions is used for the expansion: $\mathcal{L}\Phi_j(x)$ on the interior and $\mathcal{B}\Phi_j(x)$ on the boundary.

Collocating (i.e., substituting Equation 1.17 into Equations 1.13-1.15) we get:

$$\sum_{j=1}^{N_I} c_j \mathcal{L}^2\Phi_j(x_i) + \sum_{j=N_I+1}^{N} c_j \mathcal{L}\mathcal{B}\Phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{L}P_l(x_i) = f(x_i) \qquad i = 1, ..., n_I \tag{1.18}$$

$$\sum_{j=1}^{N_I} c_j \mathcal{B}\mathcal{L}\Phi_j(x_i) + \sum_{j=N_I+1}^{N} c_j \mathcal{B}^2\Phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{B}P_l(x_i) = g(x_i) \qquad i = n_I + 1, ..., n \tag{1.19}$$

$$\sum_{j=1}^{N_I} c_j \mathcal{L}P_l(x_j) + \sum_{j=N_I+1}^{N} c_j \mathcal{B}P_l(x_j) = 0 \qquad l = 1, ..., M \tag{1.20}$$

which is reformatted as the linear system:

$$\begin{bmatrix} \Phi_{\mathcal{L}\mathcal{L}} & \Phi_{\mathcal{L}\mathcal{B}} & P_{\mathcal{L}} \\ \Phi_{\mathcal{B}\mathcal{L}} & \Phi_{\mathcal{B}\mathcal{B}} & P_{\mathcal{B}} \\ P_{\mathcal{L}}^T & P_{\mathcal{B}}^T & 0 \end{bmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} f \\ g \\ 0 \end{pmatrix} \tag{1.21}$$

Note that $\Phi_{\mathcal{L}\mathcal{L}}$ represents the first summation in Equation 1.18. The linear system generated by Fasshauer's method reveals an interesting structure: namely, the subscripts $\mathcal{L}$ and $\mathcal{B}$ show blocks of influence in the matrix. For example, the interior RBF centers influence collocation on the interior collocation points ($\Phi_{\mathcal{L}\mathcal{L}}$), boundary centers influence collocation on the interior ($\Phi_{\mathcal{L}\mathcal{B}}$), interior centers influence collocation on the boundary($\Phi_{\mathcal{B}\mathcal{L}}$), and so forth. In the case where the collocation points and RBF centers do not coincide, the subscripts would also indicate which set of points the operators are applied to [**?**].

The symmetry of Fasshauer's (*symmetric collocation*) method is apparent in Equation 1.21. Likewise, it is clear that the symmetric method requires more storage and computation to solve compared to Kansa's method. However, based on the assumption that collocation points coincide with RBF centers, the symmetry reduces storage requirements by half.

1.5 Direct Collocation

In *Direct collocation* (see [**?**, **?**], the interpolant is chosen as Equation 1.3 (the same as Kansa's method). However, the Direct method collocates both the interior and boundary operators at the boundary points:

$$\sum_{j=1}^{N} c_j \mathcal{L}\Phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{L}P_l(x_i) = f(x_i) \qquad i = 1, ..., n \tag{1.22}$$

$$\sum_{j=1}^{N} c_j \mathcal{B}\Phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{B}P_l(x_i) = g(x_i) \qquad i = 1, ..., n_B = n - n_I \tag{1.23}$$

$$\sum_{j=1}^{N} c_j P_l(x_j) = 0 \qquad l = 1, ..., M \tag{1.24}$$

Reformulating as a linear system we get:

$$\begin{bmatrix} \Phi_{\mathcal{L}} & P_{\mathcal{L}} \\ \Phi_{\mathcal{B}} & P_{\mathcal{B}} \\ P^T & 0 \end{bmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} f \\ g \\ 0 \end{pmatrix} \tag{1.25}$$

While the final system in Equation 1.25 is structured the same as Kansa's method (Equation 1.16), careful inspection of the index i in Equations 1.13 and 1.22 reveals that Direct collocation produces a larger system.

1.6 Local Methods

In all three cases above, the collocation required interpolation with globally supported RBFs. However, the latest trend is to use RBFs defined with some cut-off radius to enforce compact support. In some cases, authors have used the compact support to produce a single (large) sparse system for interpolation (see e.g., [**?**, **?**, **?**, **?**, **?**]). Other approaches use compact support to produce local linear systems defined at each collocation point. Examples of this include [**?**, **?**] for Kansa's method, [**?**, **?**, **?**] for Fasshauer's method. To our knowledge no one has considered local Direct collocation. Also, instead of specifying a cut-off radius, some authors specify the exact stencil size (i.e., number of neighboring points to include); see e.g., [**?**, **?**].

After observing the general structure of the symmetric and unsymmetric collocation methods above, it is necessary only to present the symmetric (i.e. Fasshauer's) local method and note that in the unsymmetric case certain blocks will be zero allowing the system to shrink.

The formula for the interpolant local to the (k)-th collocation point (i.e., RBF center) is given by:

$$u_\Phi^{(k)}(x_i) = \sum_{j(k)=1}^{N_I} c_j^{(k)} \mathcal{L}\Phi_j(x_i) + \sum_{j(k)=N_I+1}^{N_S} c_j^{(k)} \mathcal{B}\Phi_j(x_i) + \sum_{l=1}^{M} d_l^{(k)} P_l(x_i) \quad (1.26)$$

where N_S represents the number of points that defines the local stencil; N is possibly a function of the cut-off radius in the RBF, N_I is the number of interior stencil points (those points of the stencil that lie in the interior of Ω). The index j is a function of the stencil center k allowing the system to include a local neighborhood of stencil points.

Collocating produces a linear system with similar structure to the global collocation problem, but the dimensions are much smaller:

$$\begin{bmatrix} \Phi_{\mathcal{L}\mathcal{L}} & \Phi_{\mathcal{L}\mathcal{B}} & P_{\mathcal{L}} \\ \Phi_{\mathcal{B}\mathcal{L}} & \Phi_{\mathcal{B}\mathcal{B}} & P_{\mathcal{B}} \\ P_{\mathcal{L}}^T & P_{\mathcal{B}}^T & 0 \end{bmatrix} \begin{pmatrix} c^{(k)} \\ d^{(k)} \end{pmatrix} = \begin{pmatrix} f \\ g \\ 0 \end{pmatrix} \quad (1.27)$$

Solving this system gives an interpolant locally defined around the stencil center. Note that approximating the PDE solution $u(x)$ requires finding the stencil center nearest x, then using the local interpolant for that stencil. Since interpolation is local (i.e., $c_j^{(k)}$'s are unique to each RBF center), reconstructing the derivatives with Equation 1.11 is limited to an inner product for each center rather than the matrix-vector grouping possible with global RBFs. Note that because the interpolants are local, there is no notion of global continuity/smoothness of the solution.

1.7 RBF-FD

While most of the literature surrounding RBFs for PDEs involves collocation, an alternative method does exist: RBF-FD. RBF-FD is a hybrid of RBF scat-

tered data interpolation and Finite Difference stencils.The idea behind finite-difference stencils is to express various derivative operators as a linear combination of the unknown functional values in the neighborhood of the point where an approximation to the derivative operator is desired. Common approximations such as upwind differencing, center differencing, higher order approximations, and even spectral operators, are of this form. A common approach to building such discrete operators is to form a local interpolant in a neighborhood of the target point, and simply differentiate it analytically. This is the approach taken in RBF-FD, which allows for stencils with irregular placement and number of nodes, and assigns their weights based on an RBF [**?**]. For a detailed derivation of the method see [**?**, **?**, **?**, **?**]. Such an approach leads to very simple implementations of time-advancement schemes, whether explicit or implicit. The solution at the new time step is simply some linear (if $\mathcal{L}()$ is linear), nonlinear otherwise, combination of the unknown functional values (if implicit scheme) or known functional value (if explicit scheme).

Key challenges lie in the choice of grid, the choice of stencil, whether or not to change the support as a function of the stencil, how to guaranty the stability of the differentiation operator after discretizaiton, etc.

1.8 Time Stepping

Until now we have assumed that the differential operator $\mathcal{L}$ had no time dependency. However, many important problems are a function of time:

$$\frac{\partial u(x;t)}{\partial t} = \mathcal{L}u(x;t) \tag{1.28}$$

$$\mathcal{B}(u(x;t)) = g(x) \tag{1.29}$$

with some initial condition $u(x;0) = f(x)$. Both collocation methods and RBF-FD methods can be used to discretize the right hand side. In the case of collocation, the coefficients c are now functions of time. On the other hand, it is the unknown functional values that are considered to be the unknown, time-dependent variables when RBF-FD is used.

The general approach for collocation methods is to first use the initial and boundary conditions to get an initial set of coefficients $c(t), d(t)$, whereas before, (c, d) denotes the list of unknown coefficients of the RBF basis functions and the monomials. Prior to spatial discretization, Equation 1.29 is updated using any of the standard methods for time-stepping, such as Euler, Crank-Nicholson, or higher order Runge-Kutta methods to name a few. Here we demonstrate the approach with the Euler method (which is explicit) for simplicity. In practice, a higher order method would be chosen, perhaps an implicit method. The Euler method can be written as

$$u(x;t_{n+1}) = u(x;t_n) + \Delta t(\mathcal{L}u(x;t_n)) \tag{1.30}$$

Since $u(x;0) = u(x;t_0)$ is known as a function of $(c, d) = A^{-1}u(x;t_n)$, it is straightforward to compute the coefficients, and from there, $\mathcal{L}(u(x;t_n))$.

Equation 1.30 is solved via a two step algorithm:

1. solve Equation 1.12 to approximate $\mathcal{L}u(x;t_n)$ for the current time-step
2. update $u(x;t_{n+1})$

Initially, $A_{\mathcal{L}}A^{-1}$ is computed once during the first time-step, and the first part of the algorithm is reduced to an inner product to get $\mathcal{L}u(x;t_n)$. Once $u(x;t_{n+1})$ is updated, one repeats the procedure with $u(x;t_{n+1})$ as a new initial condition. For collocation methods, one must recompute all the coefficients at each timestep (i.e. recompute A^{-1}), which is expensive. On the other hand, for a static grid, $A_{\mathcal{L}}A^{-1}$ can be precomputed and there is no longer any need to work in coefficient space. In this case, we have a type of RBF-FD method, which is solely expressed in terms of functional values. Of course, in a "pure" RBF-FD method, one has expressions for the various derivatives and operators as a linear combination of functional values (the derived stencil being an implicit function of the RBFs and their location).

There is a huge literature on time-stepping schemes for finite-difference, finite-volume, spectral methods, etc., and many of these methods can be adopted to the solve time-dependent problems on meshless grids. Although there are surely issues that are particular to RBFS, we do not discuss them here. Instead, we will include our experiences in the final thesis.

For initial value problems with moving nodes, the coefficients must be updated at each time-step whether working with collocation or RBF-FD. We do not consider moving node problems in this thesis.

Chapter 2

RBF Methods for PDE Solutions

2.1 History

The process of solving PDEs using RBFs dates back to 1990 [**?**, **?**]. This chapter starts with a description of the general approximation problem and provides background on RBF scattered data interpolation that will be required for the remainder of the chapter. This is followed by Author's Note: FINISH

We categorize existing methods for solving PDEs with RBFs as either global or local. Global methods are based on collocation and invert a single large linear system to find the interpolant that satisfies the differential equations at nodes in the domain. Local methods limit the influence of basis functions and seek an interpolant at each node defined in terms of neighboring basis functions (local collocation) or nodal values (RBF-FD).

The selling points of RBF-FD are numerous. While not a panacea for PDEs, the method combines many inherited traits of global RBF methods with lower computational complexity. As demonstrated here, the method is simple to code, easily extensible for higher accuracy and dimensions, and powerful in its ability to avoid singularities introduced by the coordinate system that might impact other methods.

2.2 Global RBF Method

Consider a PDE expressed in terms of (linear) differential operators, $\mathcal{L}$ and $\mathcal{B}$:

$$\begin{aligned} \mathcal{L}u &= f \quad \text{on } \Omega, \\ \mathcal{B}u &= g \quad \text{on } \Gamma \end{aligned} \tag{2.1}$$

where Ω is the interior of the physical domain, Γ is the boundary of Ω and f, g are known explicitly. In the case of a non-linear differential operator, a Newton's

iteration, or some other method, can be used to linearize the problem (see e.g., [**?**]); of course, this increases the complexity of a single time step.

We solve PDEs of this form with *collocation*. That is, given a number of points in the domain, we seek the derivative and solution values that best satisfy f, g at node locations. Author's Note: Not clear yet. Review Lynch2005 for a better explanation[**?**]

Global RBF collocation methods pose the problem of interpolating a multivariate function $f : \Omega \to \mathbb{R}$ where $\Omega \subset \mathbb{R}^m$. Given a set of sample values $\{f(x_j)\}_{j=1}^N$ on a discrete set of nodes $X = \{x_j\}_{j=1}^N \subset \Omega$, an approximation $\hat{f}_N$ can be constructed through linear combinations of interpolation functions. Here, we choose univariate, radially symmetric functions based on Euclidean distance ($||\cdot||$), and use translates $\phi(x - x_j)$ of a single continuous real valued function ϕ defined on $\mathbb{R}$ and centered at x_j:

$$\phi(x) := \varphi(||x||).$$

Here, φ is a Radial Basis Function and ϕ the associated kernel. For simplification $\phi_j(x)$ refers to a kernel centered at x_j; i.e., $\varphi(||x - x_j||)$.

The interpolant $\hat{f}_N(x)$ requires a linear combination of translates:

$$\hat{f}_N(x) = \sum_{j=1}^{N} c_j \phi_j(x)$$

with real coefficients $\{c_j\}_{j=1}^N$. Assuming the interpolant passes through known values of f; i.e.,

$$\hat{f}_N(x_i) = f(x_i), \qquad 1 \le i \le N,$$

allows one to solve for coefficients if the following linear system is uniquely solvable:

$$\sum_{j=1}^{N} c_j \phi_j(x_i) = f(x_i), \qquad 1 \le i \le N.$$

This is true if the $N \times N$ matrix Φ produced by the linear system

$$\begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_N(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_N(x_2) \\ \vdots & \ddots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_N(x_N) \end{pmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{bmatrix}$$

$$\Phi \vec{c} = \vec{f}$$

is nonsingular.

When choosing an appropriate basis function for interpolation, a subset of RBFs have been shown to produce symmetric positive definite Φ, while others are only conditionally positive definite (for more details see [**?**]). With the latter

set, additional polynomial terms are added to constrain the system and enforce positive definiteness, resulting in this expanded system of equations:

$$\begin{aligned}\sum_{j=1}^{N} c_j \phi_j(x_i) + \sum_{k=1}^{M} d_k p_k(x_i) &= f(x_i), \qquad 1 \le i \le N, \\ \sum_{j=1}^{N} c_j p_k(x_j) &= 0, \qquad 1 \le k \le M. \\ \begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix} \begin{bmatrix} c \\ d \end{bmatrix} &= \begin{bmatrix} \vec{f} \\ 0 \end{bmatrix} \end{aligned} \tag{2.2}$$

where $\{p_k\}_{k=1}^{M}$ is a basis for Π_p^m (the set of polynomials in m variables of degree $\le p$) and

$$M = \binom{p+m}{m}.$$

Given the coefficients $\vec{c}$, the function value at a test point x is interpolated by

$$\begin{aligned}\hat{f}_N(x) &= \sum_{j=1}^{N} c_j \Phi_j(x) + \sum_{l=1}^{M} d_l P_l(x) \\ &= \begin{bmatrix} \Phi & \mathbb{P} \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} \\ &= \vec{\Phi}_x^T \vec{c} \\ &= \vec{\Phi}_x^T \Phi^{-1} \vec{f} \end{aligned} \tag{2.3}$$

where $\vec{c}$ is substituted by the solution to Equation 2.2. In Equation 2.3 the term $\Phi_x^T \Phi^{-1}$, dependent only on node positions, can be evaluated prior to knowing f.

2.3 The RBF-generated Finite Differences Method

For the RBF-FD method, derivatives of $u(x)$ are weighted combinations of a small neighborhood of N_s nodes (i.e., a stencil with $N_s \ll N$):

$$\mathcal{L}u(x_1) \approx \sum_{j=1}^{N_s} c_j u(x_j)$$

where $\mathcal{L}u$ denotes a linear differential quantity over u (e.g., $\mathcal{L}u = \frac{du}{dx}$). Here, c_j are unknown, and obtained by enforcing the reconstruction:

$$\mathcal{L}\phi_j(x_1) = \sum_{i=1}^{N_s} c_i \phi_j(x_i) \qquad \text{for } j = 1, 2, ..., N_s$$

with $\mathcal{L}\phi_j$ provided by analytically applying the differential operator to the kernel function. This is equivalent to:

$$\begin{pmatrix} \phi_1(x_1) & \phi_1(x_2) & \cdots & \phi_1(x_{N_s}) \\ \phi_2(x_1) & \phi_2(x_2) & \cdots & \phi_2(x_{N_s}) \\ \vdots & \ddots & \ddots & \vdots \\ \phi_{N_s}(x_1) & \phi_{N_s}(x_2) & \cdots & \phi_{N_s}(x_{N_s}) \end{pmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} \mathcal{L}\phi_1(x_1) \\ \mathcal{L}\phi_2(x_1) \\ \vdots \\ \mathcal{L}\phi_{N_s}(x_1) \end{bmatrix} \tag{2.4}$$

$$\Phi_s \vec{c} = \vec{\Phi_{\mathcal{L}_s}}. \tag{2.5}$$

Solving Equation 2.5 for stencils weights, $\vec{c}$, and substituting into Equation 2.7 reveals the similarity between RBF-FD and the general problem of RBF interpolation (see Equation 2.3 and discussion):

$$\begin{aligned} \mathcal{L}u(x_1) &\approx (\Phi_s^{-1}\vec{\Phi}_{\mathcal{L}_s})^T \vec{u} \\ &\approx \vec{\Phi}_{\mathcal{L}_s}^T \Phi_s^{-T} \vec{u} \end{aligned}$$

Again, based on the choice of RBF, positive definiteness of the system is ensured by adding the same constraints as Equation 2.2, but note differential quantities for $\vec{\mathbb{P}}$ on the right hand side:

$$\begin{pmatrix} \Phi_s & \mathbb{P} \\ \mathbb{P}^T & 0 \end{pmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \vec{\Phi}_{\mathcal{L}_s} \\ \vec{\mathbb{P}}_{\mathcal{L}} \end{bmatrix}. \tag{2.6}$$

Equation 2.6 is strictly for the stencil centered at node x_1. To obtain weights for all stencils in the domain, a total of N such systems must be solved.

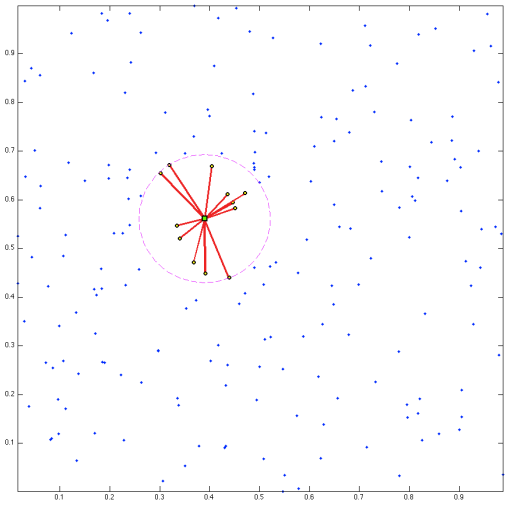
2.4 Stencil-based Derivative Approximation

The RBF-FD method is similar to classical Finite Differences in that differential quantities at a single node referred to as a *center*, $\mathbf{x}_j$, is approximated by weighted combinations of function values at $n-1$ neighboring nodes. The center node and its neighbors define a *stencil*, $\{\mathbf{x}\}_{i=1}^n$, in a localized (small) region or *neighborhood* of the domain.

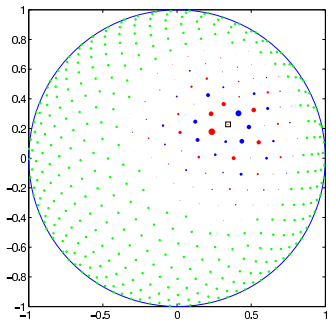
Figure 2.1 provides two examples of RBF-FD stencils. First, Figure 2.1a illustrates a single stencil of size $n = 13$ in a domain of randomly distributed nodes. The stencil center is represented by a green square, with the 12 neighboring nodes connected via red edges. The purple circle, the minimum covering circle for the stencil, demonstrates that the stencil contains only the 12 nearest neighbors of the center node. In Figure 2.1b, a larger RBF-FD stencil of size $n = 75$ on the unit sphere is shown as red and blue disks surrounding the center represented as a square. Green disks are nodes outside of the stencil. The radii and color the red and blue disks represent the magnitude and sign of the RBF-FD weights determined to calculate a derivative quantity at the stencil center.

Weights for both classical Finite Difference and RBF-FD can be obtained through the solution of linear systems. In the case of Finite Difference, the

system is a Vandermonde matrix. For RBF-FD the system is based on a symmetric distance matrix. Author's Note: Need better description and ref: The key difference between FD and RBF-FD systems is the singularity that occurs when two nodes swap.



(a) A 13 node RBF-FD stencil of randomly distributed nodes. The stencil centered at the green square contains the 12 nearest neighbors contained within the minimum covering circle drawn in purple.



(b) A 75 node RBF-FD stencil with blue (negative) and red (positive) differentiation weights to approximate advective operator at the square. Stencils weights indicated by scale of disk radii. (Image courtesy of Bengt Fornberg and Natasha Flyer)

Figure 2.1: Examples of stencils computable with RBF-FD Author's Note: Consider: show example FD stencil (5pt with regular grid)

2.5 RBF-FD weights

Given a set of function values, $\{u(\mathbf{x}_j)\}_{j=1}^N$, on a set of N nodes $\{\mathbf{x}_j\}_{j=1}^N$, the operator $\mathcal{L}$ acting on $u(\mathbf{x})$ evaluated at $\mathbf{x}_j$, is approximated by a weighted combination of function values, $\{u(\mathbf{x}_i)\}_{i=1}^n$, in a small neighborhood of $\mathbf{x}_j$, where $n \ll N$ defines the size of the stencil.

$$\mathcal{L}u(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_j} \approx \sum_{i=1}^{n} w_i u(\mathbf{x}_i) + w_{n+1} p_0 \tag{2.7}$$

The RBF-FD weights, w_i, are found by enforcing that they are exact within the space spanned by the RBFs $\phi_i(\epsilon r) = \phi(\epsilon\,||\mathbf{x} - \mathbf{x}_i||)$, centered at the nodes $\{\mathbf{x}_i\}_{i=1}^n$, with $r = ||\mathbf{x} - \mathbf{x}_i||$ being the distance between where the RBF is centered and where it is evaluated as measured in the standard Euclidean 2-norm. Various studies show [?, ?, ?, ?] that better accuracy is achieved when the interpolant can exactly reproduce a constant, p_0. Assuming $p_0 = 1$, the constraint

$\sum_{i=1}^{n} w_i = \mathcal{L}1|_{\mathbf{x}=\mathbf{x}_j} = 0$ completes the system:

$$\begin{pmatrix} \phi(\epsilon\,||\mathbf{x}_1-\mathbf{x}_1||) & \phi(\epsilon\,||\mathbf{x}_1-\mathbf{x}_2|| & \cdots & \phi(\epsilon\,||\mathbf{x}_1-\mathbf{x}_n||) & 1 \\ \phi(\epsilon\,||\mathbf{x}_2-\mathbf{x}_1||) & \phi(\epsilon\,||\mathbf{x}_2-\mathbf{x}_2|| & \cdots & \phi(\epsilon\,||\mathbf{x}_2-\mathbf{x}_n||) & 1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ \phi(\epsilon\,||\mathbf{x}_n-\mathbf{x}_1||) & \phi(\epsilon\,||\mathbf{x}_n-\mathbf{x}_2|| & \cdots & \phi(\epsilon\,||\mathbf{x}_n-\mathbf{x}_n||) & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ w_{n+1} \end{bmatrix} = \begin{bmatrix} \mathcal{L}\phi(\epsilon\,||\mathbf{x}-\mathbf{x}_1||)|_{\mathbf{x}=\mathbf{x}_j} \\ \mathcal{L}\phi(\epsilon\,||\mathbf{x}-\mathbf{x}_2||)|_{\mathbf{x}=\mathbf{x}_j} \\ \vdots \\ \mathcal{L}\phi(\epsilon\,||\mathbf{x}-\mathbf{x}_n||)|_{\mathbf{x}=\mathbf{x}_j} \\ 0 \end{bmatrix}, \tag{2.8}$$

where w_{n+1} is ignored after the matrix in (2.8) is inverted. This $n \times n$ system solve is repeated for each stencil center $\mathbf{x}_j$, $j = 1...N$, to form the N rows of the DM with n non-zeros ($n \ll N$) per row. As an example, if $\mathcal{L}$ is the identity operator, then the above procedure leads to RBF-FD interpolation. If $\mathcal{L} = \frac{\partial}{\partial x}$, one obtains the DM that approximates the first derivative in x. In the context of time-dependent PDEs, the stencil weights remain constant for all time-steps when the nodes are stationary. Therefore, the calculation of the differentiation weights is performed once in a single preprocessing step of $O(n^3 N)$ FLOPs. Improved efficiency is achieved by processing multiple right hand sides in one pass, calculating the weights corresponding to all required derivative quantities (i.e., $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$, ∇^2, etc.).

For each of the N small system solves of Equation (2.8), the n nearest neighbors to $\mathbf{x}_j$ need to be located. This can be done efficiently using neighbor query algorithms or spatial partitioning data-structures such as Locality Sensitive Hashing (LSH) and kD-Tree. Different query algorithms often have a profound impact on the DM structure and memory access patterns. We choose a Raster (ijk) ordering LSH algorithm [**?**] leading to the matrix structure in Figures **??** and **??**. While querying neighbors for each stencil is an embarrassingly parallel operation, the node sets used here are stationary and require stencil generation only once. Efficiency and parallelism for this task has little impact on the overall run-time of tests, which is dominated by the time-stepping. We preprocess node sets and generate stencils serially, then load stencils and nodes from disk at run-time. In contrast to the RBF-FD view of a static grid, Lagrangian/particle based PDE algorithms promote efficient parallel variants of LSH in order to accelerate querying neighbors at each time-step [**?**, **?**].

2.6 Weight Operators

Throughout the development of our parallel code we have verified code correctness through the solution of a variety of PDEs. Here we provide a list of operators we have tested and their corresponding equations for the RHS of Equation 2.8 necessary to compute RBF-FD weights.

The standard first derivatives $\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}$ are produced by the chain rule

$$\frac{\partial \phi}{\partial x} = \frac{dr}{dx}\frac{d\phi}{dr} = \frac{(x - x_j)}{r}\frac{d\phi}{dr} \tag{2.9}$$

$$\frac{\partial \phi}{\partial y} = \frac{dr}{dy}\frac{d\phi}{dr} = \frac{(y - y_j)}{r}\frac{d\phi}{dr} \tag{2.10}$$

$$\frac{\partial \phi}{\partial z} = \frac{dr}{dz}\frac{d\phi}{dr} = \frac{(z - z_j)}{r}\frac{d\phi}{dr} \tag{2.11}$$

where $\frac{\partial \phi}{\partial r}$ for the Gaussian RBFs is given by:

2.6.1 Laplacian (∇^2)

2D:

3D:

2.6.2 Laplace-Beltrami (Δ_S) on the Sphere

The ∇^2 operator can be represented in spherical polar coordinates for $\mathbb{R}^3$ as:

$$\nabla^2 = \underbrace{\frac{1}{r}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right)}_{\text{radial}} + \underbrace{\frac{1}{r^2}\Delta_S}_{\text{angular}}, \tag{2.12}$$

where Δ_S is the Laplace-Beltrami operator—i.e., the Laplacian operator constrained to the surface of the sphere. This form nicely illustrates the separation of components into radial and angular terms.

In the case of PDEs solved on the unit sphere, there is no radial term, so we have:

$$\nabla^2 \equiv \Delta_S. \tag{2.13}$$

Although this originated in the spherical coordinate system, [**?**] introduced the following Laplaci-Beltrami operator for the surface of the sphere:

$$\Delta_S = \frac{1}{4}\left[\left(4 - r^2\right)\frac{\partial^2}{\partial r^2} + \frac{4 - 3r^2}{r}\frac{\partial}{\partial r}\right], \tag{2.14}$$

where r is the Euclidean distance between nodes of an RBF-FD stencil and is independent of our choice of coordinate system.

2.6.3 Constrained Gradient ($P_x \cdot \boldsymbol{\nabla}$) on the Sphere

Additionally following [**?**, **?**], the gradient operator must also be constrained to the sphere with this projection matrix:

$$P = I - \mathbf{x}\mathbf{x}^T = \begin{pmatrix} (1 - x_1^2) & -x_1x_2 & -x_1x_3 \\ -x_1x_2 & (1 - x_2^2) & -x_2x_3 \\ -x_1x_3 & -x_2x_3 & (1 - x_3^2) \end{pmatrix} = \begin{pmatrix} P_{x_1} \\ P_{x_2} \\ P_{x_3} \end{pmatrix} \tag{2.15}$$

where $\mathbf{x}$ is the unit normal at the stencil center.

The direct method of computing RBF-FD weights for the projected gradient for $\mathbf{P} \cdot \nabla$ is presented in [**?**]. When solving for the weights, we apply the projection on the right hand side of our small linear system. We let $\mathbf{x} = (x_1, x_2, x_3)$ be the stencil center, and $\mathbf{x}_k = (x_{1,k}, x_{2,k}, x_{3,k})$ indicate an RBF-FD stencil node.

Using the chain rule, and assumption that

$$r(\mathbf{x}_k - \mathbf{x}) = ||\mathbf{x}_k - \mathbf{x}|| = \sqrt{(x_{1,k} - x_1)^2 + (x_{2,k} - x_2)^2 + (x_{3,k} - x_3)^2},$$

we obtain the unprojected gradient of ϕ as

$$\nabla\phi(r(\mathbf{x}_k - \mathbf{x})) = \frac{\partial r}{\partial \mathbf{x}} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r} = -(\mathbf{x}_k - \mathbf{x}) \frac{1}{r(\mathbf{x}_k - \mathbf{x})} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}$$

.

Applying the projection matrix gives

$$\begin{aligned}
\mathbf{P}\nabla\phi(r(\mathbf{x}_k - \mathbf{x})) &= -(\mathbf{P} \cdot \mathbf{x}_k - \mathbf{P} \cdot \mathbf{x}) \frac{1}{r(\mathbf{x}_k - \mathbf{x})} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r} \\
&= -(\mathbf{P} \cdot \mathbf{x}_k - 0) \frac{1}{r(\mathbf{x}_k - \mathbf{x})} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r} \\
&= -(I - \mathbf{x}\mathbf{x}^T)(\mathbf{x}_k) \frac{1}{r(\mathbf{x}_k - \mathbf{x})} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r} \\
&= \begin{pmatrix} x\mathbf{x}^T\mathbf{x}_k - x_k \\ y\mathbf{x}^T\mathbf{x}_k - y_k \\ z\mathbf{x}^T\mathbf{x}_k - z_k \end{pmatrix} \frac{1}{r(\mathbf{x}_k - \mathbf{x})} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}
\end{aligned}$$

Thus, we directly compute the weights for $P_x \cdot \boldsymbol{\nabla}$ using these three RHS in Equation 2.8:

$$P\frac{\partial}{\partial x_1} = (x_1\mathbf{x}^T\mathbf{x}_k - x_{1,k}) \frac{1}{r(\mathbf{x}_k - \mathbf{x})} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}|_{\mathbf{x}=\mathbf{x}_j} \tag{2.16}$$

$$P\frac{\partial}{\partial x_2} = (x_2\mathbf{x}^T\mathbf{x}_k - x_{2,k}) \frac{1}{r(\mathbf{x}_k - \mathbf{x})} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}|_{\mathbf{x}=\mathbf{x}_j} \tag{2.17}$$

$$P\frac{\partial}{\partial x_3} = (x_3\mathbf{x}^T\mathbf{x}_k - x_{3,k}) \frac{1}{r(\mathbf{x}_k - \mathbf{x})} \frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}|_{\mathbf{x}=\mathbf{x}_j} \tag{2.18}$$

Alternatively, assuming weights for operators $\boldsymbol{\nabla} = \frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_3}$ are already computed, the projected operator could be constructed via weighting the unprojected gradient components. For example:

Author's Note: Show accuracy of derivative approximation when using projected operator vs linear combinations of dx,dy,dz operators

2.7 Stabilization: Hyperviscosity

For RBF-FD, differentiation matrices encode convective operators of the form

$$D = \alpha \frac{\partial}{\partial \lambda} + \beta \frac{\partial}{\partial \theta} \tag{2.19}$$

where α and β are a function of the fluid velocity. The convective operator, discretized through RBF-FD, has eigenvalues in the right half-plane causing the method to be unstable [**?**, **?**]. Stabilization of the RBF-FD method is achieved through the application of a hyperviscosity filter to Equation (2.19) [**?**]. By using Gaussian RBFs, $\phi(r) = e^{-(\epsilon r)^2}$, the hyperviscosity (a high order Laplacian operator) simplifies to

$$\Delta^k \phi(r) = \epsilon^{2k} p_k(r) \phi(r) \tag{2.20}$$

where k is the order of the Laplacian and $p_k(r)$ are multiples of generalized Laguerre polynomials that are generated recursively (see [**?**]: Section 3.2). We assume a 2D Laplacian operator when working on the surface of the sphere since a local stencil can be viewed as lying on a plane.

In the case of parabolic and hyperbolic PDEs, hyperviscosity is added as a filter to the right hand side of the evaluation. For example, at the continuous level, the equation solved takes the form

$$\frac{\partial u}{\partial t} = -Du + Hu, \tag{2.21}$$

where D is the PDE operator, and H is the hyperviscosity filter operator. Applying hyperviscosity shifts all the eigenvalues of D to the left half of the complex plane. This shift is controlled by k, the order of the Laplacian, and a scaling parameter γ_c, defined by

$$H = \gamma \Delta^k = \gamma_c N^{-k} \Delta^k.$$

Given a choice of ϵ (see Section **??**), it was found experimentally that $\gamma = \gamma_c N^{-k}$ provides stability and good accuracy for all values of N considered here. It also ensures that the viscosity vanishes as $N \to \infty$ [**?**]. In general, the larger the stencil size, the higher the order of the Laplacian. This is attributed to the fact that, for convective operators, larger stencils treat a wider range of modes accurately. As a result, the hyperviscosity operator should preserve as much of that range as possible. The parameter γ_c must also be chosen with care and its sign depends on k (for k even, γ_c will be negative and for k odd, it will be positive). If γ_c is too large, the eigenvalues move outside the stability domain of our time-stepping scheme and/or eigenvalues corresponding to lower physical modes are not left intact, reducing the accuracy of our approximation. If γ_c is too small, eigenvalues remain in the right half-plane [**?**, **?**].

Part II

The RBF-FD Method

Chapter 3

The RBF-FD Method

This is followed by a description of Radial Basis Function-generated Finite Differences and examples of operators approximated by the method within this work.

The choice to study RBF-FD within this dissertation is motivated by two factors. First, RBF-FD represents one of the latest developments within the RBF community. The method was formally introduced in 2003 but has yet to obtain the critical-mass following necessary for the method's use in large-scale scientific models. Our goal throughout the dissertation has been to scale RBF-FD to complex problems on high resolution meshes, and to lead the way for its adoption in high performance computational geophysics. Second, RBF-FD inherits many of the positive features from global and local collocation schemes, but sacrifices others for the sake of significantly reduced computational complexity and increased parallelism. We capitalize on the inherent parallelism to develop a collection of multi-GPU test cases that span the compute nodes of a Top 500 supercomputer. Our effort leads the way for application of RBF-FD in the modern field of high performance computing where GPU accelerators will be key to breaching the exa-scale barrier in computing [**?**].

Rather than solely focus on the optimizations of said algorithms on the GPU, we dedicate significant attention to the practical application of RBF-FD to interesting problems in geophysics. This means we have walked a fine line between research topics to both apply the method to

Chapter 4

TEMP

4.1 Fragments (integrate above)

Stabilization of the RBF-FD method is achieved through the application of a hyperviscosity filter [**?**]. By assuming the use of Gaussian RBFs, $\phi(r) = e^{-(\epsilon r)^2}$, the hyperviscosity operator simplifies to

$$\Delta^k \phi(r) = \epsilon^{2k} p_k(r) \phi(r). \tag{4.1}$$

The multiples of generalized Laguerre polynomials, $p_k(r)$, are obtained through the following recursive relation:

$$\begin{cases} p_0(r) &= 1, \\ p_1(r) &= 4(\epsilon r)^2 - 2d, \\ p_k(r) &= 4((\epsilon r)^2 - 2(k-1) - \frac{d}{2}) p_{k-1}(r) - 8(k-1)(2(k-1) - 2 + d) p_{k-2}(r), \qquad k = 2, 3, ... \end{cases}$$

where d is the dimension of the problem. We assume $d = 2$ below when working on the surface of the sphere.

Many algorithms exist to query the k-nearest neighbors (equivalently all nodes in the minimum/smallest enclosing circle). Some algorithms overlay a grid similar to Locality Sensitive Hashing and query such as... [**?**].

Conditioning

Conditioning is defined as:

With condition number estimates we can choose a proper support parameter for uniformly distributed node sets.

Alternative algorithms exist for solving for RBF-FD weights when the systems are overly ill-conditioned. Currently, we have an unpublished algorithm ContourSVD available to play with (demonstrate accuracy improvements).

Node Placement – Centroidal Voronoi Tessellation

We make the assumption for now that we use regularly distributed nodes. Centroidal Voronoi tessellations provide reasonably good distributions for solutions. Examples (heat ellipse, ellipsoid, sphere, square cavity).

The motivation behind RBF-FD is generality/functionality in the numerical method. Scattered nodes are supported. Distribution requires proper choice of support, and tight nodes result in increased conditioning

4.2 Approximate Nearest Neighbor (ANN) Query

RBF methods are traditionally described as general and meshless in that they apply to unstructured clouds of points in arbitrary dimensions. However, although the term meshless implies a method capable of operating with no node connectivity, all numerical methods—meshless RBF methods included—connect nodes in the domain. For example, the "meshless" global RBF method connects every node in the domain to all other nodes. Compact support or local RBF methods like RBF-FD limit connections to nodes that lie within a predetermined radius.

The connections between nodes form a directed adjacency graph with edges that dictate the paths along which data/phenomena can travel. For example, a plus shaped stencil of five points with a center node and four neighboring nodes allows values to propagate north, south, east and west; not northeast, southeast, etc.

They are robust and function on scattered point clouds. RBF-FD in particular requires stencils to be generated from n nearest neighbors to a stencil center. The cost of these neighbor queries can vary greatly depending on the choice of algorithm or data-structure used to make the query.

For example, in general brute force is inefficient The author of [**?**] queries n nearest neighbors for a compact-support RBF partition of unity example with a k-D tree. In [**?**, **?**] a k-D Tree is leveraged for all neighbor queries for RBF-FD.

In our work in [**?**] an alternative to k-D tree was leveraged, based loosely on Locality Sensitive Hashing.

4.2.1 k-D Tree

Most of the RBF community leverages the k-D tree, due to its low computational complexity for querying neighbors and its wide availability as standalone software in the public domain (e.g., matlab central has a few implementations for download, and the MATLAB Statistics Toolbox includes an efficient k-D Tree).

The complexity of assembling he tree is

The Matlab central k-D Tree is MEX compiled and efficient. We integrated the standalone C++ code into our library.

While the k-D Tree functions well for queries, its downfall is a large cost in preprocessing to build the tree. For moving nodes, such as in Lagrangian

schemes, this cost is prohibitively high. In an attempt to reduce the cost, lagrangian schemes introduced approximate nearest neighbor queries based on

Approximate nearest neighbors will be nearly balanced. We observe that RBF-FD functions as well on stencils of true nearest neighbors as it does on approximate nearest neighbors.