UNIONE EUROPEA

Fondo Sociale Europeo

UNIVERSITÀ DEGLI STUDI DI TORINO

DIPARTIMENTO DI MATEMATICA

DOTTORATO DI RICERCA IN MODELING AND DATA SCIENCE

CICLO: 37°

TITOLO DELLA TESI: OPTIMIZATION TECHNIQUE FOR MESHFREE APPROXIMATION METHODS

TESI PRESENTATA DA: SANDRO LANCELLOTTI

SUPERVISORE: PROF.SSA ALESSANDRA DE ROSSI

COORDINATORE DEL DOTTORATO: PROF.SSA ROSA MEO

ANNI ACCADEMICI: 2021-2022, 2022-2023, 2023-2024, 2024-2025

SETTORE SCIENTIFICO-DISCIPLINARE DI AFFERENZA: MAT/08

Progetto svolto grazie al sostegno finanziario del **Programma Operativo Nazionale Ricerca e Innovazione 2014-2020 (CCI2014IT16M2OP005), risorse FSE REACT-EU, Azione IV.5 "Dottorati su tematiche Green"**

Contents

List of Tables

List of Figures

Introduction

Kernel-based methods are widely acknowledged as powerful tools for the approximation and interpolation of scattered data. They are particularly notable in approximation theory and meshfree methods due to their ability to handle high-dimensional problems while maintaining accuracy. These methods are based on weighted combinations of radial kernels, often referred to as Radial Basis Functions (RBFs) [43]. RBF methods have established themselves as adaptable and robust tools in meshfree techniques, contributing in solving a variety of problems across scientific and engineering domains [10, 30, 51, 109]. Their appeal lies in their simplicity for high-dimensional implementations, flexibility in handling complex geometries, and favorable convergence behavior [133]. However, these methods often result in full and computationally expensive linear systems that may also suffer from ill-conditioning. To address these challenges, attention has shifted to a meshfree strategy known as the RBF partition of unity method (RBF-PUM). This approach uses local RBF approximants and combines their outputs summing them in a weighted way through a global partition of unity scheme.

The partition of unity scheme, which has its origins in Shepard's method [121], was designed to reconstruct functions from scattered data. By breaking the original problem into smaller, localized subproblems, this method has found applications in a range of computational mathematics and scientific computing fields [5, 22, 27, 29]. The combination of RBF interpolation with PUM was first explored by Wendland [131], which also provided error analysis for functions within the native RBF space.

The performance in RBF methods is influenced by a key parameter known as the *shape parameter*. The effectiveness of radial kernel techniques depends heavily on this parameter, making its proper selection a pivotal task. As noted by Wendland [133], a balance must be maintained between numerical stability and accuracy. Historically, the shape parameter has often been chosen using trial-and-error approaches, which can be both time-intensive and computationally demanding. Alternatively, Franke [53] and Hardy [60] proposed specific criteria, though these are not always optimal. Systematic strategies for determining an ideal parameter value or range have been introduced in Biazar et al. [11] and Scheuerer [114], offering broad applicability but often incurring significant computational costs. Selecting an appropriate scale parameter remains a topic of active research in various areas of applied mathematics and computational science (e.g., [22, 32, 69]).

Beyond applications to scattered data, PUMs can be extended to datasets with intrinsic structures, such as data defined on graph vertices. Signal processing on graph addresses problems associated with such structured data, including tasks such as filtering, compression, noise reduction, sampling, and decomposition [90, 124]. Graphs frequently arise in

real-world applications, including social networks and traffic systems [27]. However, the structural complexity of graphs often necessitates sophisticated analytical methods for efficient processing.

The partition of unity method offers a robust framework for addressing tasks such as signal reconstruction, node classification, and localized filtering on specific graph segments [27]. By leveraging localized computations, it enables the reconstruction of the global signal through a systematic combination of these local solutions. When combined with approximation technique based on generalized translates of a graph basis function (GBF), the partition of unity method achieves notable gains in computational efficiency by producing sparser system matrices for interpolation and collocation tasks. This improvement not only accelerates computations but also reduces resource demands. However, the overall accuracy of this approach is closely tied to the quality of the identified communities within the graph [41, 42].

A key feature of real-world graphs is their modular structure, marked by densely connected groups and relatively sparse links between them. Identifying these clusters is vital in fields like sociology, biology, and computer science, where graphs are widely used as modeling tools. Despite its importance, community detection remains a challenging and unsolved problem, requiring ongoing research to refine methods and improve outcomes [50].

This thesis focuses on two pivotal topics: optimizing the hyperparameters for Radial Basis Function (RBF) interpolation using Bayesian Optimization (BO) and developing a topology-driven approach for optimal community detection in Graph-Based Functions with Partition of Unity Methods (GBF-PUM). Specifically, BO is applied to improve the efficiency of shape parameter tuning in RBF interpolation, demonstrating comparable accuracy to the classical Leave-One-Out Cross-Validation (LOOCV) method while significantly reducing computational costs. In the RBF-PUM framework, BO is extended for simultaneous optimization of the shape parameter and subdomain radius, offering a feasible alternative for scenarios where computational demands are high. Furthermore, in the GBF-PUM context, a novel community detection technique leverages the graph's topology to define subdomains, ensuring that each contains at least one interpolation node, addressing signal interpolation problems on graph vertices.

In chapter 1 we introduce the interpolation problem and provide theoretical foundations for RBF approximation. Topics such as positive definite functions, error bounds in their native space, stability, and the trade-off between accuracy and stability inherent in these methods are covered.

In chapter 2 the focus is on the Partition of Unity (PU) method. We delve into the error bounds introduced by the decomposition and present practical implementations of the algorithm in both MATLAB and Python.

In chapter 3 we examine the optimization techniques employed in this thesis. Specifically, we discuss LOOCV, the traditional approach for parameter tuning in RBF interpolation, and Bayesian Optimization, a statistical technique recently adapted to the context of RBF, offering a novel contribution in this area.

In chapter 4 we apply BO to optimize the shape parameter in RBF interpolation and approximation problems. The performances are compared against LOOCV, demonstrating that BO achieves comparable accuracy with significantly lower computational costs.

In chapter 5 we extend the optimization approach to the RBF-PUM framework. BO is employed for the simultaneous optimization of the shape parameter and the subdomain

radius. In this chapter BO is proposed as a feasible alternative for RBF-PUM, particularly in scenarios where LOOCV becomes computationally prohibitive.

In chapter 6 the focus shifts to graph-based interpolation. After introducing fundamental concepts of the theory and the problem of signal interpolation on the vertices, we focus on a novel topology-driven technique for community detection in GBF-PUM. This method leverages the graph's topology to define subdomains for the PU method while ensuring each subdomain contains at least one interpolation node.

Through these investigations, the thesis exposes innovative methods for parameter optimization and community detection, contributing to the broader fields of interpolation, approximation theory, and graph-based methods.

Chapter 1

Radial Basis Functions interpolation

The first chapter is a collection of the main results in the field of interpolation using radial basis functions relied upon in the manuscript. The reference text on which the reader can find further insights and unaddressed topics related to the subject is the work of G. E. Fasshauer [43].

1.1 The Problem of Scattered Data Interpolation

Meshfree methods have become increasingly popular in the field of Approximation Theory due to their ability to handle complex problems and high-dimensional data more effectively than traditional numerical methods. These methods are particularly advantageous because they do not rely on a predefined mesh, thereby eliminating the computational expense associated with mesh generation. This makes them a faster and more powerful set of numerical tools suitable for a wide range of applications across various scientific area. A particular and widespread problem in this context is the multivariate scattered data interpolation. This involves finding a function that fits given data points, ensuring the function matches the measurements at specific locations. This process, known as interpolation, is especially relevant when the data points do not lie on a regular grid, leading to the concept of scattered data interpolation. Formally the Scattered Data Interpolation is the following:

Problem 1.1.1 (**scattered data interpolation**). *Let* $\Omega \subseteq \mathbb{R}^m$, $m \geq 1$, $X_n = \{\mathbf{x}_i | \mathbf{x}_i \in \Omega, i = 1, \dots n\}$ *and* $F_n = \{f_i | f_i = f(\mathbf{x}_i), f_i \in \mathbb{R}, i = 1, \dots n\}$ *where* f *is an unknown function. Given a (finite) set of data* $(\mathbf{x}_i, f_i)$, $i = 1, \dots, n$, *the scattered data interpolation problem consists of determining a (continuous) function* $I : \Omega \to \mathbb{R}$ *such that* $I(\mathbf{x}_i) = f_i$, $i = 1, \dots, n$.

The $\mathbf{x}_i$ are the *data points*, *nodes* or *data sites*, the (corresponding) f_i are the *data values* or *function values* and the function I can be write as a linear combination of certain *basis functions* B_j, i.e.,

$$I(\mathbf{x}) = \sum_{j=1}^{n} a_j B_j(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^m. \tag{1.1}$$

whose coefficients can be found solving the associated linear system of the form

$$Ac = f,$$

where the entries of the *interpolation matrix* A are defined as $A_{ij} = B_j(\mathbf{x}_i)$ for $i, j = 1, \ldots, n$, $c = [c_1, c_2, \ldots, c_n]^T$, and $f = [f_1, \ldots, f_n]^T$.

If the matrix A is non singular the Problem 1.1.1 is well posed, in other words, the solution of the linear system exists and is unique. The 1.1.1 is presented in the multivariate space where, unlike in the 1-dimensional space where an arbitrary set of n points can be interpolate uniquely by a $n-1$-degree polynomial, the theorem 1.1.3 shows that the that the uniqueness of the interpolant can only be achieved by making the basis functions B_j dependent on the data points. To present the theorem 1.1.3, we need the definition of the Haar spaces.

Definition 1.1.2. The finite-dimensional linear space $\mathcal{B} \subseteq C(\Omega)$ with basis $\{B_1, B_2, \ldots, B_n\}$ is a Haar space on Ω if

$$\det A \neq 0,$$

for any set of distinct data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ in Ω.

The invertibility of the interpolation matrix A and hence the existence and the uniqueness of the interpolant (1.1), are attained if the basis functions form a Haar space. The simplest example of a Haar space is the set of all $n-1$-degree univariate polynomials that is a Haar space of dimension n for the points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ within the space $\mathcal{B}$

Thus, a Haar space ensures the invertibility of the interpolation matrix A, guaranteeing both the existence and uniqueness of an interpolant in the form (1.1). Additionally, it is well-established that univariate polynomials of degree $n-1$ constitute an n-dimensional Haar space for data points located at $\mathbf{x}_1, \ldots, \mathbf{x}_n$ within the space $\mathcal{B}$. This property doesn't scale to higher dimensional space (see [71, 38] for more details) making the search for a unique interpolant more difficult.

Theorem 1.1.3 (**Mairhuber-Curtis**). *Suppose that $\Omega \subset \mathbb{R}^m$, $m \geq 2$, contains an interior point, then all and only Haar spaces are those generated by a single function.*

As mentioned earlier, the only way for the basis functions to form a Haar space, ensuring the well-posedness of problem 1.1.1, is by making the basis functions B_j dependent on the data points.

1.1.1 Interpolation by Radial Basis Function

Radial Basis Functions (RBF) are a class of functions whose values depend only on the norm of their argument. This means that the value of these functions is constant at all points equidistant from the origin, hence the term *radial* or *spherical*. They are widely used as basis functions because the RBF interpolant remains invariant under Euclidean transformations (translation, rotation, and reflection), allowing these transformations to be applied before or after calculating the interpolant. Notably, interpolation using RBFs is independent of the dimensionality of the space n because these functions depend only on the magnitude of the data points' norms.

Definition 1.1.4. A function $\Phi : \mathbb{R}^m \to \mathbb{R}$ is called *radial* provided that exists a *univariate* function $\varphi : [0, \infty) \to \mathbb{R}$ such that

$$\Phi(\mathbf{x}) = \varphi(r),$$

where $r = \|\mathbf{x}\|$, and $\|\cdot\|$ is some norm on $\mathbb{R}^m$, usually the Euclidean norm.

The previous definition can be translated as follows:

$$\|\mathbf{x}_1\| = \|\mathbf{x}_2\| \Longrightarrow \Phi(\mathbf{x}_1) = \Phi(\mathbf{x}_2), \qquad \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^m.$$

Choosing RBF as basis functions the interpolant 1.1 can be written as follows:

$$I(\mathbf{x}) = \sum_{j=1}^{n} a_j \varphi(\|\mathbf{x} - \mathbf{x}_j\|_2), \qquad \mathbf{x} \in \mathbb{R}^m. \tag{1.2}$$

Now, imposing the interpolation condition of the problem 1.1.1

$$I(\mathbf{x}_i) = f_i, \qquad i = 1, \dots, n$$

we obtain the linear system

$$\begin{bmatrix} \varphi(|\mathbf{x}_1 - \mathbf{x}_1|_2) & \varphi(|\mathbf{x}_1 - \mathbf{x}_2|_2) & \cdots & \varphi(|\mathbf{x}_1 - \mathbf{x}_n|_2) \\ \varphi(|\mathbf{x}_2 - \mathbf{x}_1|_2) & \varphi(|\mathbf{x}_2 - \mathbf{x}_2|_2) & \cdots & \varphi(|\mathbf{x}_2 - \mathbf{x}_n|_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(|\mathbf{x}_n - \mathbf{x}_1|_2) & \varphi(|\mathbf{x}_n - \mathbf{x}_2|_2) & \cdots & \varphi(|\mathbf{x}_n - \mathbf{x}_n|_2) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}.$$

whose solution leads to the determination of the coefficients $a_1, \dots, a_n$. The theorem 1.1.3 states that the well-posedness of the problem 1.1.1 is linked to the choice of the basis functions and thus, we are interested if radial basis functions brings to a non singular solution of the previous system. To show that it always occurs, we introduce the notion of positive definite matrices and positive definite functions.

1.2 Positive definite matrices and functions

The characterization of the class of functions that generate well-posed problems, as outlined in problem 1.1.1, remains an active area of research. Nonetheless, focusing on positive definite functions provides simpler and more easily verifiable results, ensuring a non-singular interpolation matrix. A comprehensive discussion on positive definite functions and the interpolation of scattered data is available in [133]. To address the necessity of a non-singular interpolation matrix, this section introduces the theoretical concepts and results related to positive definite functions.

1.2.1 Positive definite matrices

Definition 1.2.1. A real symmetric matrix $\boldsymbol{A}$ is called *positive semi-definite* if its associated quadratic form is non-negative, i.e.,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j A_{ij} \geq 0 \tag{1.3}$$

for $\mathbf{c} = [c_1, c_2, \ldots, c_n]^T \in \mathbb{R}^n$. If the quadratic form (1.3) is zero only for $c \equiv 0$, then A is called *positive definite*.

Positive definite matrices have positive eigenvalues, ensuring they are non-singular (though the converse is not necessarily true). Therefore, if the basis functions B_j for $j = 1, 2, \ldots, n$ are chosen such that they generate a positive definite interpolation matrix for any set of distinct data points, the problem 1.1.1 will be well-posed. A common approach to obtaining a positive definite interpolation matrix is to choose shifts of a certain function centered at $\mathbf{x}_j$ as the basis functions. In other words, $B_j(\cdot) = \Phi(\cdot - \mathbf{x}_j)$. To this end, we introduce *positive definite functions*.

1.2.2 Positive definite functions

Definition 1.2.2. A complex-valued continuous function $\Phi : \mathbb{R}^m \to \mathbb{C}$ is called *positive definite* on $\mathbb{R}^m$ if

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i \overline{c_j} \Phi\left(\mathbf{x}_i - \mathbf{x}_j\right) \geq 0 \tag{1.4}$$

for any n pairwise different data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^m$, and $\mathbf{c} = [c_1, c_2, \ldots, c_n]^T \in \mathbb{C}^n$. The function Φ is called *strictly positive definite* on $\mathbb{R}^m$ if the quadratic form (1.4) is zero only for $\mathbf{c} \equiv 0$.

Here, we present an extended definition encompassing complex-valued functions and complex coefficients $\mathbf{c}$, facilitating the utilization of techniques such as Fourier transforms in a more natural manner. This treatment proves advantageous when deriving certain properties of (strictly) positive definite functions. Additionally, the renowned *Bochner's theorem* (see Theorem 1.2.8) precisely characterizes the positive definite functions defined in Definition 1.2.2. However, it becomes evident that for even, real-valued functions, it suffices to examine the quadratic form solely for real values $\mathbf{c} \in \mathbb{R}^n$.

During the 20th century, positive definite functions were introduced into the field of mathematical analysis, first by Mathias [76] in the 1920s, who introduced them and studied their properties. A comprehensive collection of results and developments on positive definite functions up to the 1970s can be found in [125]. Positive definite functions have been defined in analogy to positive semi-definite matrices. To achieve a well-posed interpolation problem, it is necessary to refine the classical notion of a positive definite function to that of a strictly positive definite function. The first to study strictly positive definite functions was Micchelli [80], who also established a connection between scattered data interpolation and strictly positive definite functions. Additional insights into this topic can be found in the works of Schoenberg [116, 117].

Thus, a good approach is to consider strictly positive definite function as a basis functions in (1.1) to obtain the well-posedness of the problem 1.1.1.

$$I(\mathbf{x}) = \sum_{j=1}^{n} c_j \Phi\left(\mathbf{x} - \mathbf{x}_j\right), \quad \mathbf{x} \in \mathbb{R}^m.$$

Even though strictly positive definite functions are defined in the complex field, we are primarily interested in real data problems. This definition aids in identifying properties of

strictly positive definite functions and implies that only those functions whose quadratic form is real are candidates for strictly positive definite functions. The theorem 1.2.3 shows some basic properties of positive definite functions.

Theorem 1.2.3. *Let Φ a positive definite function. The following properties hold:*

(1) $\Phi(\mathbf{0}) \geq 0$.

(2) $\Phi(-\mathbf{x}) = \overline{\Phi(\mathbf{x})}$.

(3) Any positive definite function is bounded. In fact,

$$|\Phi(\mathbf{x})| \leq \Phi(\mathbf{0}).$$

(4) If Φ is positive definite with $\Phi(0) = 0$, then $\Phi \equiv 0$.

(5) The product of (strictly) positive definite functions is (strictly) positive definite.

(6) Non-negative finite linear combinations of positive definite functions are positive definite. If $\Phi_1, \Phi_2, \ldots, \Phi_n$ are positive on $\mathbb{R}^m$ and $c_j \geq 0$, $j = 1,2,\ldots,n$, then

$$\Phi(\mathbf{x}) = \sum_{j=1}^{n} c_j \Phi_j(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^m,$$

is also positive definite. Moreover, if at least one of the Φ_j is strictly positive definite and the corresponding $c_j > 0$, then Φ is strictly positive definite.

From definition 1.2.2 and property (2) follow the characterization Theorem for real-valued (strictly) positive definite functions.

Theorem 1.2.4. *A real-valued continuous function Φ is positive definite on $\mathbb{R}^m$ if and only if it is even and*

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \Phi(\mathbf{x}_i - \mathbf{x}_j) \geq 0 \tag{1.5}$$

for any n pairwise different data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^m$, and $c = [c_1, c_2, \ldots, c_n]^T \in \mathbb{R}^n$. The function Φ is called strictly positive definite on $\mathbb{R}^m$ if the quadratic form (1.5) is zero only for $\mathbf{c} \equiv 0$.

Additional insights about strictly positive definite functions and their characterization in the real field can be found in [133].

1.2.3 Integral characterizations

Integral characterization of positive definite functions were firstly studied and formalized in the 1930s by Bochner and Schoenberg . Notably, Bochner's theorem is paramount in the theoretical treatment of RBFs. Since the Fourier transform and some properties are used to state Bochner's theorem let us introduce them.

Definition 1.2.5. The Fourier transform of $f \in L_1(\mathbb{R}^m)$ is defined by

$$\hat{f}(\boldsymbol{\omega}) = \frac{1}{\sqrt{(2\pi)^m}} \int_{\mathbb{R}^m} f(\mathbf{x}) e^{-i\boldsymbol{\omega}\cdot\mathbf{x}} dx, \quad \boldsymbol{\omega} \in \mathbb{R}^m, \tag{1.6}$$

and its inverse Fourier transform by

$$\check{f}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^m}} \int_{\mathbb{R}^m} f(\boldsymbol{\omega}) e^{i\mathbf{x}\cdot\boldsymbol{\omega}} d\boldsymbol{\omega}, \quad \mathbf{x} \in \mathbb{R}^m.$$

Definition 1.2.6. Let μ a finite (signed) measure on $\mathbb{R}^m$. The Fourier transform of μ is defined by

$$\hat{\mu}(\boldsymbol{\omega}) = \frac{1}{\sqrt{(2\pi)^m}} \int_{\mathbb{R}^m} e^{-i\boldsymbol{\omega}\cdot\mathbf{x}} d\mu(\mathbf{x}), \quad \boldsymbol{\omega} \in \mathbb{R}^m.$$

An important property of radial function, stated by theorem 1.2.7 is that radiality is invariant under Fourier transformation, i.e. the Fourier transform of a radial function is radial.

Theorem 1.2.7. *Let $\Phi \in L_1(\mathbb{R}^m)$ be continuous and radial, i.e. $\Phi(\mathbf{x}) = \varphi(\|\mathbf{x}\|)$. Then its Fourier transform $\hat{\Phi}$ is also radial, i.e. $\hat{\Phi}(\boldsymbol{\omega}) = \mathcal{F}_m\varphi(\|\boldsymbol{\omega}\|)$ with*

$$\mathcal{F}_m\varphi(r) = \frac{1}{\sqrt{r^{m-2}}} \int_0^\infty \varphi(t) t^{m/2} J_{(m-2)/2}(rt) dt,$$

where $J_{(m-2)/2}$ is the classical Bessel function of the first kind of order $(m-2)/2$.

$\mathcal{F}_m\varphi(r)$ is often called the *Hankel transform* or the *Fourier-Bessel transform*. For further details about special functions see [54]. Moreover the *Hankel inversion theorem* [122] asserts that the compositions of the Fourier transform for radial functions with itself act as the identity. Thus for a given radial function φ

$$\mathcal{F}_m\left[\mathcal{F}_m\varphi\right] = \varphi.$$

The highest achievements in the characterization of positive definite functions in terms of Fourier transforms was established for $m = 1$ [13] and for general m [14] by Bochner, in 1932 and 1933 respectively.

Theorem 1.2.8 (Bochner's theorem). *A (complex-valued) function $\Phi \in C(\mathbb{R}^m)$ is positive definite on $\mathbb{R}^m$ if and only if it is the Fourier transform of a finite non-negative Borel measure μ on $\mathbb{R}^m$, i.e.*

$$\Phi(\mathbf{x}) = \hat{\mu}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^m}} \int_{\mathbb{R}^m} e^{-i\mathbf{x}\cdot\mathbf{y}} d\mu(\mathbf{y}), \quad \mathbf{x} \in \mathbb{R}^m.$$

Given our objective of ensuring a well-posed interpolation problem, our focus shifts towards extending Bochner's characterization to *strictly* positive definite functions.

With the aim of extending Bochner's characterization to *strictly* positive definite functions to ensure the well-posedness of the interpolation problem let us introduce the notion of carrier:

Definition 1.2.9. Let μ be a non-negative finite Borel measure on a topological space $\mathbf{x}$, the carrier of μ is

$$\mathbf{x} \setminus \bigcup \{O \mid O \text{ is open and } \mu(O) = 0\}.$$

The ensuing result furnishes a sufficient condition for a function to be strictly positive definite on $\mathbb{R}^m$. The following theorem enounces a sufficient condition for the strict positive definiteness of a function.

Theorem 1.2.10. *Let μ be a non-negative finite Borel measure on $\mathbb{R}^m$ whose carrier is a set of nonzero Lebesgue measure. Then the Fourier transform of μ is strictly positive definite on $\mathbb{R}^m$.*

As a corollary to the previous theorem, we have a corollary that explains how to construct functions

Corollary 1.2.11. *Let $\Phi \in L_1(\mathbb{R}^m)$ be a continuous non-negative function, not identically zero. Then the Fourier transform of Φ is strictly positive definite on $\mathbb{R}^m$.*

Finally, a criterion for determining if a given function is strictly positive definite can be found in [133]. The last theorem of this section (see [133]), is a validation for strict positive definiteness of functions.

Theorem 1.2.12. *A continuous function $\Phi \in L_1(\mathbb{R}^m)$ is strictly positive definite if and only if Φ is bounded and its Fourier transform is non-negative and not identically equal to zero.*

Theorem 1.2.12 is crucial in the study of strict positive definiteness. Assuming Φ is not identically zero (which implies that $\hat{\Phi}$ is also not identically zero), for Φ to be strictly positive definite, it suffices that $\hat{\Phi}$ be non-negative.

1.2.4 Positive definite radial functions

Definition 1.2.2 pertains to multivariate functions Φ that are (strictly) positive definite. However, if Φ is radial, $\Phi(\mathbf{x}) = \varphi(\|\mathbf{x}\|)$, we can refer to φ as a *positive definite radial function* even if it represents an abuse of terminology widely used in literature.

With the previous convention, we can expose an important result for strictly positive definite radial functions.

Lemma 1.2.13. *Suppose that $\Phi = \varphi(\|\cdot\|)$ is (strictly) positive definite and radial on $\mathbb{R}^m$, then Φ is also (strictly) positive definite and radial on $\mathbb{R}^p$ with $p \leq m$.*

The following theorem, due to Schoenberg [115], characterize positive definite radial functions in terms of integral representation.

Theorem 1.2.14. *A continuous function $\varphi : [0, \infty) \to \mathbb{R}$ is positive definite and radial on $\mathbb{R}^m$ if and only if it is the Bessel transform of a finite non-negative Borel measure μ on $[0, \infty)$, i.e.*

$$\varphi(r) = \int_0^\infty \Omega_m(rt) d\mu(t),$$

where

$$\Omega_m(r) = \begin{cases} \cos r, & \text{for } m = 1, \\ \Gamma\left(\frac{m}{2}\right)\left(\frac{2}{r}\right)^{(m-2)/2} J_{(m-2)/2}(r), & \text{for } m \geq 2, \end{cases}$$

and $J_{(m-2)/2}$ is the classical Bessel function of the first kind of order $(m-2)/2$.

In [133], Wendland characterizes strictly positive definite radial functions on $\mathbb{R}^m$ through the use of the Fourier transform. This characterization integrates Theorem 1.2.12 with the formula presented in Theorem 1.2.7 for the Fourier transform of a radial function.

Theorem 1.2.15. *A continuous function $\varphi : [0, \infty) \to \mathbb{R}$ such that $r \mapsto r^{m-1}\varphi(r) \in L_1[0, \infty)$ is strictly positive definite and radial on $\mathbb{R}^m$ if and only if the m-dimensional Fourier transform*

$$\mathcal{F}_m\varphi(r) = \frac{1}{\sqrt{r^{m-2}}} \int_0^\infty \varphi(t) t^{m/2} J_{(m-2)/2}(rt) dt$$

is non-negative and not identically equal to zero.

A characterization of positive definite functions'class on $\mathbb{R}^m$ for any dimension m is due to Schoenberg [115].

Theorem 1.2.16 (Schoenberg). *A continuous function $\varphi : [0, \infty] \to \mathbb{R}$ is strictly positive definite and radial on $\mathbb{R}^m$ for all m if and only if it is of the form*

$$\varphi(r) = \int_0^\infty e^{-r^2 t^2} d\mu(t),$$

where μ is a finite non-negative Borel measure on $[0, \infty)$ not concentrated at the origin.

To find a zero r_0 of φ, by Theorem 1.2.16 we have to solve

$$\varphi(r_0) = \int_0^\infty e^{-r_0^2 t^2} d\mu(t) = 0.$$

Nevertheless, since μ is non-negative and the exponential function is positive, μ must be the zero measure, making φ identically zero. This implies that no positive definite and radial function φ can have zeros, except for the trivial function $\varphi \equiv 0$.

Theorem 1.2.17. *There are no oscillatory univariate continuous functions that are strictly positive definite and radial on $\mathbb{R}^m$ for all m. Moreover, there are no compactly supported univariate continuous functions that are strictly positive definite and radial on $\mathbb{R}^m$ for all m.*

An example of a strictly positive definite radial function is the Gaussian function.

$$\Phi(\mathbf{x}) = e^{-\varepsilon^2 \|\mathbf{x}\|^2}, \qquad \varepsilon > 0,$$

For any m it is strictly positive definite (and radial) on $\mathbb{R}^m$ because the Fourier transform of a Gaussian remains a Gaussian.

$$\hat{\Phi}(\boldsymbol{\omega}) = \frac{1}{(\sqrt{2}\varepsilon)^m} e^{\frac{-\|\boldsymbol{\omega}\|^2}{4\varepsilon^2}}.$$

The parameter ε is known as the *shape parameter* for its ability to modify the shape of the function. Notably, small values of theε generate flatter basis functions while high epsilon values result in sharper functions. Moreover, setting $\varepsilon = 1/\sqrt{2}$, we have $\hat{\Phi} = \Phi$.

1.2.5 Completely monotone functions

Completely monotone functions represents a class of functions closely linked to positive definite radial functions. The introduction of completely monotone functions lead to a simple and effective characterization of positive definite radial functions.

Definition 1.2.18. A function $\varphi : [0, \infty) \to \mathbb{R}$ is called *completely monotone* on $[0, \infty)$ if $\varphi \in C[0, \infty) \cap C^{\infty}(0, \infty)$ and

$$(-1)^l \varphi^{(l)}(r) \geq 0, \qquad r > 0, \qquad l = 0,1,2,\ldots \tag{1.7}$$

A more thorough examination of the properties can be found in [133].

Some important properties of completely monotone functions [134, 47**?**], that are worth mentioning, are as follows:

(A) A finite linear combination of completely monotone functions with non-negative coefficients is completely monotone.

(B) Complete monotonicity is preserved in the product of completely monotone functions.

(C) The composition of an absolutely monotone function ϕ (i.e., $\phi^{(l)} \geq 0$ for all $l \geq 0$) and a completely monotone function φ, $\phi \circ \varphi$, is completely monotone.

(D) If ψ is a positive function whose derivative is completely monotone and φ is completely monotone, then $\varphi \circ \psi$ is completely monotone.

We now present a significant result concerning the integral characterization of completely monotone functions. This characterization, which will be crucial for establishing the connection with strictly positive definite radial functions, was independently studied by Bernstein (1914 and 1928), Haussdorff (1921), and Widder (1931).

Theorem 1.2.19 (**Bernstein-Hausdorff-Widder**). *A function $\varphi : [0, \infty) \to \mathbb{R}$ is completely monotone on $[0, \infty)$ if and only if it is the Laplace transform of a finite non-negative Borel measure μ on $[0, \infty)$, i.e. φ is of the form*

$$\varphi(r) = \mathcal{L}\mu(r) = \int_0^{\infty} e^{-rt} d\mu(t).$$

The actual connection between positive definite radial and completely monotone functions is exposed in the following theorem due to Schoenberg (1938).

Theorem 1.2.20. *A function φ is completely monotone on $[0, \infty)$ if and only if $\Phi = \varphi(\|\cdot\|^2)$ is positive definite and radial on $\mathbb{R}^m$ for all m.*

The following theorem, due to Schoenberg [115] provide a well-posedness test for problems related to the interpolation of scattered data. It states that completely monotonicity implies strict positive definiteness. Also the converse is true and a proof can be found in [133].

Theorem 1.2.21. *A function $\varphi : [0, \infty) \to \mathbb{R}$ is completely monotone but not constant if and only if $\varphi(\|\cdot\|^2)$ is strictly positive definite and radial on $\mathbb{R}^m$ for all m.*

Gaussian function is completely monotone on $[0,\infty)$; in particular, we have that $\varphi(r) = e^{-\varepsilon r}$, $\varepsilon \geq 0$, is completely monotone on $[0,\infty)$ since

$$(-1)^l \varphi^{(l)}(r) = \varepsilon^l e^{-\varepsilon r} \geq 0, \qquad l = 0,1,2,\ldots;$$

Moreover, since the gaussian function is not constant, Theorem 1.2.21 guarantee that $\Phi(\mathbf{x}) = \varphi(\|\mathbf{x}\|^2) = e^{-\varepsilon^2\|\mathbf{x}\|^2}$, with $\varepsilon > 0$, is strictly positive definite and radial on $\mathbb{R}^m$ for any m.

1.2.6 Multiply monotone functions

Monotonicity can be used as a test also to verify if a radial function on $\mathbb{R}^m$ for a given fixed value of m is strictly positive definite. To this aim, we introduce the notion of *multiply monotone functions*.

Definition 1.2.22. Let $\varphi : (0,\infty) \to \mathbb{R}$ with $\varphi \in C^{k-2}(0,\infty)$ for $k \geq 2$ and $(-1)^l \varphi^{(l)}(r)$ is non-negative, non-increasing, and convex for each $l = 0,\ldots,k-2$. Then, φ is called *k-times monotone* on $(0,\infty)$. If $k = 1$ we only require $\varphi \in C(0,\infty)$ to be non-negative and non-increasing.

As for completely monotone function, to build a relationship between strictly positive definite radial functions and multiply monotone functions, an integral representation is needed. This representation was determined by Williamson [135].

Theorem 1.2.23 (Williamson). *A continuous function $\varphi : (0,\infty) \to \mathbb{R}$ is k-times monotone on $(0,\infty)$ if and only if it is of the form*

$$\varphi(r) = \int_0^\infty (1-r)_+^{k-1} d\mu(t),$$

where μ is a non-negative Borel measure on $(0,\infty)$.

The following theorem proposed by Micchelli [80] and refined by Buhmann [8], actually instantiates the connection between strictly positive definite radial functions and multiply monotone functions.

Theorem 1.2.24 (Micchelli). *Let $k = \lfloor m/2 \rfloor + 2$ be a positive integer. Suppose that $\varphi : [0,\infty) \to \mathbb{R}$, $\varphi \in C[0,\infty)$, is k-times monotone on $(0,\infty)$ but not constant, then φ is strictly positive definite and radial on $\mathbb{R}^m$ for any m such that $\lfloor m/2 \rfloor \leq k-2$.*

The theories of completely monotone and multiply monotone functions are quite similar but exhibit some key differences. Notably, if φ is completely monotone and not constant, then by Theorem 1.2.21, $\varphi(\cdot^2)$ is strictly positive definite on $\mathbb{R}^m$ for any m. However, in the case of multiply monotone functions, the square term is not included in Theorem 1.2.24.

Furthermore, the equivalence between completely monotone functions and strictly positive definite radial functions on $\mathbb{R}^m$ for any m, as guaranteed by Theorem 1.2.21, does not hold for multiply monotone functions. Consequently, the converse of Theorem 1.2.24 is not true for multiply monotone functions.

1.3 Compactly supported radial basis functions

In general, a function is not strictly positive definite and radial on $\mathbb{R}^m$ for all m (see Theorem 1.2.17 and [43, Theorem 9.4]). However, there is a process to construct such functions for some fixed m. In this section, we will discuss the construction of Wendland's compactly supported functions. To do this, we recall that, according to Theorem 1.2.8, a function is strictly positive definite and radial on $\mathbb{R}^m$ if its m-variate Fourier transform is non-negative.

Furthermore, by Theorem 1.2.7, the Fourier transform of a radial function $\Phi = \varphi(\|\cdot\|)$ is also a radial function. Specifically, it is given by

$$\hat{\Phi}(\mathbf{x}) = \mathcal{F}_m\varphi(\|\mathbf{x}\|) = \|\mathbf{x}\|^{-(m-2)/2} \int_0^\infty \varphi(t) t^{m/2} J_{(m-2)/2}(t\|\mathbf{x}\|)\, dt,$$

where $J_{(m-2)/2}$ is the Bessel function of the first kind of order $(m-2)/2$.

1.3.1 Montée operator for radial functions

Definition 1.3.1. Let φ be such that $t \mapsto t\varphi(t) \in L_1[0,\infty)$. Then the *integral (montée) operator* $\mathcal{I}$ is defined by

$$(\mathcal{I}\varphi)(r) = \int_r^\infty t\varphi(t)dt, \qquad r \geq 0.$$

The resulting function should be interpreted as an even function by considering its even extension. This operator is crucial in the construction of Wendland's compactly supported radial functions. It was first proposed by Matheron [75] and thoroughly analyzed by Schaback and Wu [113] for radial functions. For a detailed treatment of the properties of this and other operators, refer to [132, 113].

Theorem 1.3.2. *The operator $\mathcal{I}$ has the following properties:*

(a) Let φ compactly supported , then $\mathcal{I}\varphi$ is compactly supported.

(b) If $t \mapsto t^{m-1}\varphi(t) \in L_1[0,\infty)$ and $m \geq 3$, then $\mathcal{F}_m(\varphi) = \mathcal{F}_{m-2}(\mathcal{I}\varphi)$.

The operator $\mathcal{I}$ allows to express m-variate Fourier transform into $(m-2)$-variate Fourier transform. In addition, from the previous properties and theorem of characterization of positive definite radial functions in terms of integral representation 1.2.14, the following theorem is established.

Theorem 1.3.3. *Suppose that $\varphi \in C(\mathbb{R})$. If $t \mapsto t^{m-1}\varphi(t) \in L_1[0,\infty)$ and $m \geq 3$, then φ is strictly positive definite and radial on $\mathbb{R}^m$ if and only if $\mathcal{I}\varphi$ is strictly positive definite and radial on $\mathbb{R}^{m-2}$.*

In light of the previous theorem, it is possible to generate new strictly positive definite radial functions from existing ones by applying the so-called "dimension-walk" technique.

1.3.2 Wendland's compactly supported functions

To apply the dimension walk technique and generate a family of strictly positive definite radial functions, we introduce a base compactly supported radial function called *truncated power function*.
The truncated power function can be expressed as follows:

$$\varphi_s(r) = (1-r)_+^s$$

According to Theorem 1.2.17, the function is not strictly positive definite on $\mathbb{R}^m$ for all m. However, for a given s such that $s \geq \lfloor m/2 \rfloor + 1$, it is strictly positive definite.
The function $(\mathbf{x})_+$, known as the *cutoff function*, is defined as follows:

$$(x)_+ = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

Starting from the truncated power function Wendland, in [132], constructed a family of strictly positive definite radial functions applying the dimension-walk technique walking through multivariate Euclidean spaces in even-numbered dimensions by applying repeatedly the operator $\mathcal{I}$ (see also [130]).

Definition 1.3.4. With $\varphi_s(r) = (1-r)_+^s$ we define

$$\varphi_{m,k} = \mathcal{I}\varphi_{\lfloor m/2 \rfloor + k + 1}.$$

Follows that $\varphi_{m,k}$ have support in the interval [0,1], and have there a polynomial representation.

Theorem 1.3.5. *The functions $\varphi_{m,k}$ are strictly positive definite and radial on $\mathbb{R}^m$ and are of the form*

$$\varphi_{m,k}(r) = \begin{cases} p_{m,k}(r), & r \in [0,1], \\ 0, & r > 1, \end{cases}$$

with a univariate polynomial $p_{m,k}$ of degree $\lfloor m/2 \rfloor + 3k + 1$. Moreover, $\varphi \in C^{2k}(\mathbb{R})$ are unique up to a constant factor, and the polynomial degree is minimal for given space dimension m and smoothness $2k$.

The theorem states that any other strictly positive radial compactly supported polynomial function that is C^{2k} on $\mathbb{R}^s$ has a higher polynomial degree.

Wendland introduce systematic method trough recursive formulas to retrieve $\varphi_{m,k}$ for all values of m and k. Here we present a short list that can be found in [44] for some cases.

Theorem 1.3.6. *The functions $\varphi_{m,k}$, $k = 0,1,2,3$, have the form*

$$\begin{aligned}
\varphi_{m,0} &= (1-r)_+^s, \\
\varphi_{m,1} &\doteq (1-r)_+^{s+1} \left[(s+1)\, r + 1\right], \\
\varphi_{m,2} &\doteq (1-r)_+^{s+2} \left[(s^2+4s+3)\, r^2 + (3s+6)\, r + 3\right], \\
\varphi_{m,3} &\doteq (1-r)_+^{s+3} \left[(s^3+9s^2+23s+15)\, r^3 + (6s^2+36s+45)\, r^2 + (15s+45)\, r + 15\right],
\end{aligned}$$

where $s = \lfloor m/2 \rfloor + k + 1$, and the symbol $\doteq$ denotes equality up to a positive constant factor.

1.4 Error bounds for strictly positive definite radial functions interpolation

In this section, we will discuss the theoretical aspects of strictly positive definite radial functions interpolation giving as a result some bound estimation of it. A thorough treatment of the topic can be found in [133].

1.4.1 Reproducing kernel Hilbert spaces

Definition 1.4.1. Let $\mathcal{H}$ be a real Hilbert space of functions $f : \Omega \to \mathbb{R}$, $\Omega \subseteq \mathbb{R}^m$, with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. A function $K : \Omega \times \Omega \to \mathbb{R}$ is called *reproducing kernel* for $\mathcal{H}$ if

1) $K(\cdot, \mathbf{x}) \in \mathcal{H}$ for all $\mathbf{x} \in \Omega$;

2) $f(\mathbf{x}) = \langle f, K(\cdot, \mathbf{x}) \rangle_{\mathcal{H}}$ for all $f \in \mathcal{H}$ and all $\mathbf{x} \in \Omega$.

The reproducing property 2 of the previous definition gives the name to the reproducing kernel that is unique for each space. Moreover, the existence is equivalent for the functional $\delta_{\mathbf{x}} \in \mathcal{H}^*$ to be bounded and linear on Ω. The space $\mathcal{H}^*$, consisting of all bounded linear functionals on $\mathcal{H}.$, is called *dual* space of $\mathcal{H}$.

Some important properties of reproducing kernels are listed below.

Theorem 1.4.2. *Suppose that $\mathcal{H}$ is a Hilbert space of functions $f : \Omega \to \mathbb{R}$ with reproducing kernel K. Then we have*

(1) $K(\mathbf{x}, \mathbf{y}) = \langle K(\cdot, \mathbf{x}), K(\cdot, \mathbf{y}) \rangle_{\mathcal{H}} = \langle \delta_{\mathbf{x}}, \delta_{\mathbf{y}} \rangle_{\mathcal{H}^*}$ *for* $\mathbf{x}, \mathbf{y} \in \Omega$*;*

(2) $K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x})$ *for* $\mathbf{x}, \mathbf{y} \in \Omega$*;*

(3) If f_n converges in Hilbert space norm to f then f_n converge to f pointwise, i.e. $\|f - f_n\|_{\mathcal{H}} \to 0$ for $n \to \infty$ then $|f(\mathbf{x}) - f_n(\mathbf{x})| \to 0$ for all $\mathbf{x} \in \Omega$.

Note that since the reproducing kernel K is positive definite, we can enhance Definition 1.2.2 by replacing $\Phi(\mathbf{x}_i - \mathbf{x}_j)$ with $K(\mathbf{x}_i, \mathbf{x}_j)$, thereby introducing the concept of a positive definite kernel.

The following theorem shows the relationship between reproducing kernel Hilbert spaces and strictly positive definite functions and kernels.

Theorem 1.4.3. *Suppose that $\mathcal{H}$ is a reproducing kernel Hilbert function space with reproducing kernel $K : \Omega \times \Omega \to \mathbb{R}$. Then K is positive definite. Moreover, K is strictly positive definite if and only if the point evaluation functionals $\delta_{\mathbf{x}}$ are linearly independent in $\mathcal{H}^*$.*

The theorem ascertains that given a reproducing kernel within Hilbert spaces we have that this kernel is also positive definite. The following step will be to flip the connection and retrieve, given a strictly positive definite basisi function, reproducing kernel Hilbert space (RKHS).

1.4.2 Native spaces

In this section, we will show that each strictly positive definite function is associated with a reproducing kernel Hilbert space, known as the *native space* for the strictly positive definite function. The Definition 1.4.1 tell us that if a function is of the form

$$f = \sum_{i=1}^{n} c_i K(\cdot, \mathbf{x}_i),$$

where $\mathbf{x}_i \in \Omega$. Than it is contained $\mathcal{H}$.

Moreover, Theorem 1.4.2 ascertain

$$\begin{aligned} \|f\|_{\mathcal{H}}^2 &= \langle f, f \rangle_{\mathcal{H}} \\ &= \left\langle \sum_{i=1}^{n} c_i K(\cdot, \mathbf{x}_i), \sum_{j=1}^{n} c_j K(\cdot, \mathbf{x}_j) \right\rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \left\langle K(\cdot, \mathbf{x}_i), K(\cdot, \mathbf{x}_j) \right\rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(\mathbf{x}_i, \mathbf{x}_j). \end{aligned}$$

We define the linear space

$$H_K(\Omega) = \operatorname{span}\{K(\cdot, \mathbf{y}) : \ \mathbf{y} \in \Omega\}$$

with the inner product $\langle \cdot, \cdot \rangle_K$ defined by

$$\left\langle \sum_{i=1}^{n} c_i K(\cdot, \mathbf{x}_i), \sum_{j=1}^{n} d_j K(\cdot, \mathbf{x}_j) \right\rangle_K = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i d_j K(\mathbf{x}_i, \mathbf{x}_j).$$

Theorem 1.4.4. *If $K : \Omega \times \Omega \to \mathbb{R}$ is a symmetric strictly positive definite kernel, then the bilinear form $\langle \cdot, \cdot \rangle_K$ defines an inner product on $H_K(\Omega)$. Furthermore, $H_K(\Omega)$ is a pre-Hilbert space with reproducing kernel K.*

The previous theorem states that $H_K(\Omega)$ is a pre-Hilbert, it doesn't need do be complete, then, we define the *native space* $\mathcal{N}_K(\Omega)$ of the kernel K as the completion of $H_K(\Omega)$ under the norm $\|\cdot\|_K$ so that $\|f\|_K = \|f\|_{\mathcal{N}_K(\Omega)}$ for every $f \in H_K(\Omega)$. For further information see [133].

A characterization of the native space can be achieved using Fourier transforms when we consider strictly positive definite (translation-invariant) functions $\Phi(\mathbf{x} - \mathbf{y}) = K(\mathbf{x}, \mathbf{y})$ with $\Omega = \mathbb{R}^m$.

Theorem 1.4.5. *Suppose that $\Phi \in C(\mathbb{R}^m) \cap L_1(\mathbb{R}^m)$ is a real-valued strictly positive definite function. Define*

$$\mathcal{G} = \left\{ f \in L_2(\mathbb{R}^m) \cap C(\mathbb{R}^m) : \ \frac{\hat{f}}{\sqrt{\hat{\Phi}}} \in L_2(\mathbb{R}^m) \right\}$$

and equip this space with the bilinear form

$$\langle f, g\rangle_{\mathcal{G}} = \frac{1}{\sqrt{(2\pi)^m}} \left\langle \frac{\hat{f}}{\sqrt{\hat{\Phi}}}, \frac{\hat{g}}{\sqrt{\hat{\Phi}}} \right\rangle_{L_2(\mathbb{R}^m)} = \frac{1}{\sqrt{(2\pi)^m}} \int_{\mathbb{R}^m} \frac{\hat{f}(\boldsymbol{\omega})\overline{\hat{g}(\boldsymbol{\omega})}}{\hat{\Phi}(\boldsymbol{\omega})} d\boldsymbol{\omega}.$$

Then $\mathcal{G}$ is a real Hilbert space with inner product $\langle\cdot,\cdot\rangle_{\mathcal{G}}$ and reproducing kernel $\Phi(\cdot - \cdot)$. Hence, $\mathcal{G}$ is the native space of Φ on $\mathbb{R}^m$, i.e. $\mathcal{G} = \mathcal{N}_\Phi(\mathbb{R}^m)$, and both inner products coincide. In particular, every $f \in \mathcal{N}_\Phi(\mathbb{R}^m)$ can be recovered from its Fourier transform $\hat{f} \in L_1(\mathbb{R}^m) \cap L_2(\mathbb{R}^m)$.

Theorem 1.4.5 establishes that the native spaces of translation-invariant functions is an extension of the framework of standard *Sobolev spaces*. Notably, for $s > m/2$, the Sobolev space W_2^s is defined as follows (see, e.g., [1]):

$$W_2^s(\mathbb{R}^m) = \left\{ f \in L_2(\mathbb{R}^m) \cap C(\mathbb{R}^m) : \hat{f}(\cdot)\left(1 + \|\cdot\|_2^2\right)^{s/2} \in L_2(\mathbb{R}^m) \right\}.$$

If a strictly positive definite function Φ possesses a Fourier transform that decays algebraically, its native space can be identified with a Sobolev space. For example, the native spaces associated with Wendland's compactly supported functions $\Phi_{m,k} = \varphi_{m,k}(\|\cdot\|_2)$ are given by $\mathcal{N}_{\Phi_{m,k}}(\mathbb{R}^m) = W_2^{m/2+k+1/2}(\mathbb{R}^m)$. It should be noted that for $k = 0$, the condition $m \geq 3$ must be met.

Theorem 1.4.5 also indicates that for Gaussian functions, the Fourier transform of any function $f \in \mathcal{N}_\Phi(\Omega)$ must exhibit faster decay than that of the Gaussian's Fourier transform. Although the native space for Gaussian functions is relatively small, it encompasses the set of *band-limited functions*, which are characterized by having compactly supported Fourier transforms.

1.4.3 Native space error estimates

The aim of this section is to determine error bounds for interpolation using strictly positive definite functions expressed in terms of the *fill distance*

$$h = h_{\mathcal{S}_n,\Omega} = \sup_{\mathbf{x}\in\Omega} \min_{\mathbf{x}_i\in\mathcal{S}_n} \|\mathbf{x} - \mathbf{x}_i\|_2, \tag{1.8}$$

that is a measure of how well the domain Ω is filled by the set of nodes $\mathcal{S}_n$ as $h \to 0$. In other words, the fill distance represents the maximum radius of an empty ball that can fit inside Ω.

Trough the results in this section we will be able to express the error estimates between a function $f \in \mathcal{N}_\Phi$ and its interpolant F in term of fill distance. To do so we can express the interpolant in the *Lagrange form* using the so called *cardinal basis functions*, the idea is due to Wu and Schaback [136].

Given Φ, strictly positive definite, the linear system

$$Ac = f$$

has a unique solution, where $A_{ij} = \Phi(\mathbf{x}_i - \mathbf{x}_j)$ for $i, j = 1, \ldots, n$, with $c = [c_1, c_2, \ldots, c_n]^T$ and $f = [f_1, \ldots, f_n]^T$. It naturally extends to the strictly positive definite kernels where $A_{ij} = \Phi(\mathbf{x}_i, \mathbf{x}_j)$.

Cardinal basis functions u_j^*, $j = 1, 2, \ldots, n$, which satisfy $u_j^*(\mathbf{x}_i) = \delta_{ij}$, i.e.

$$u_j^*(\mathbf{x}_i) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases}$$

can be determined by solving the linear system

$$Au^*(\mathbf{x}) = b(\mathbf{x}), \tag{1.9}$$

where A is invertible, $u^* = [u_1^*, u_2^*, \ldots, u_n^*]^T$, and $b = [\Phi(\cdot, \mathbf{x}_1), \Phi(\cdot, \mathbf{x}_2), \ldots, \Phi(\cdot, \mathbf{x}_n)]^T$.

Theorem 1.4.6. *Suppose that Φ is a strictly positive definite kernel on $\mathbb{R}^m$. Then, for any distinct data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, there exist functions $u_j^* \in span\{\Phi(\cdot, \mathbf{x}_j), j = 1,2,\ldots,n\}$, such that $u_j^*(\mathbf{x}_i) = \delta_{ij}$.*

Then, we can express the interpolant I, in the cardinal form

$$I(\mathbf{x}) = \sum_{j=1}^{n} f(\mathbf{x}_j) u_j^*(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^m.$$

Note that the the cardinal functions are independent from the function's values at the interpolation nodes. The cardinale functions are determined by solving the linear system (1.9) after the identification of the interpolation nodes and the choice of the optimal shape parameter.

Definition 1.4.7. Suppose that $\Omega \subseteq \mathbb{R}^m$ and $\Phi \in C(\Omega \times \Omega)$ is strictly positive definite on $\mathbb{R}^m$. For any set $\mathcal{S}_n = \{\mathbf{x}_i, i = 1,2,\ldots,n\} \subseteq \Omega$ of distinct data points the *power function* is defined by

$$[P_{\Phi,\mathcal{S}_n}(\mathbf{x})]^2 = Q\left(u^*(\mathbf{x})\right),$$

where, for any vector $u \in \mathbb{R}^m$, Q is of the form

$$\begin{aligned} Q(u) &= \Phi(\mathbf{x}, \mathbf{x}) - 2\sum_{j=1}^{n} u_j(\mathbf{x})\Phi(\mathbf{x}, \mathbf{x}_j) + \sum_{i=1}^{n}\sum_{j=1}^{n} u_i(\mathbf{x})u_j(\mathbf{x})\Phi(\mathbf{x}_i, \mathbf{x}_j) \\ &= \left\| \Phi(\cdot, \mathbf{x}) - \sum_{j=1}^{n} u_j(\mathbf{x})\Phi(\cdot, \mathbf{x}_j) \right\|^2_{\mathcal{N}_\Phi(\Omega)}, \end{aligned} \tag{1.10}$$

and u^* is the vector of cardinal functions from Theorem 1.4.6.

The following theorem exploits the previous results to obtain a first generic error estimate.

Theorem 1.4.8. *Let $\Omega \subseteq \mathbb{R}^m$, $\Phi \in C(\Omega \times \Omega)$ be strictly positive definite on $\mathbb{R}^m$, and suppose that $\mathcal{S}_n = \{\mathbf{x}_i, i = 1,2,\ldots,n\}$ is a set of distinct data points. If we denote by I the interpolant to $f \in \mathcal{N}_\Phi(\Omega)$ on $\mathcal{S}_n$, then for all $\mathbf{x} \in \Omega$ we have*

$$|f(\mathbf{x}) - I(\mathbf{x})| \leq P_{\Phi,\mathcal{S}_n}(\mathbf{x}) \, \|f\|_{\mathcal{N}_\Phi(\Omega)}.$$

Theorem 1.4.8 allows the evaluation of the interpolation error throughout:

- the data smoothness, quantified by the native space norm of f. This norm is independent of the locations of the data points but is influenced by the choice of the kernel Φ;
- the distribution of the nodes and the specific kernel Φ.

By Theorem 1.4.8, the shape parameter ε influences the native space norm of f. Thus, modifying ε results in changing the error estimation.

Our upcoming objective is to enhance this error estimate by linking the influence of the nodes to the fill distance, subsequently applying this improved estimate to different basis functions Φ.

In numerical analysis, a common approach for deriving error bounds involves leveraging the local polynomial accuracy of a method and then employing a Taylor series expansion.

Theorem 1.4.9. *Let $\Omega \subseteq \mathbb{R}^m$, and suppose that $\Phi \in C(\Omega \times \Omega)$ is strictly positive definite on $\mathbb{R}^m$. Let $\mathcal{S}_n = \{\mathbf{x}_i, i = 1,2,\dots,n\}$ be a set of distinct data points in Ω, and define the quadratic form $Q(u)$ as in (1.10). The minimum of $Q(u)$ is given by the vector $u = u^*(\mathbf{x})$ in Theorem 1.4.6, i.e.*

$$Q\left(u^*(\mathbf{x})\right) \leq Q(u) \qquad \text{for all } u \in \mathbb{R}^n.$$

Let us introduce the concept of a domain that satisfies an interior cone condition.

Definition 1.4.10. A region $\Omega \subseteq \mathbb{R}^m$ satisfies an *interior cone condition* if there exists an angle $\theta \in (0, \pi/2)$ and a radius $r > 0$ such that for all $\mathbf{x} \in \Omega$ a unit vector $\boldsymbol{\xi}(\mathbf{x})$ exists such that the cone

$$C = \left\{\mathbf{x} + \lambda\mathbf{y} : \ \mathbf{y} \in \mathbb{R}^m, \|\mathbf{y}\|_2 = 1, \mathbf{y}^T\boldsymbol{\xi}(\mathbf{x}) \geq \cos\theta, \ \lambda \in [0, r]\right\}$$

is contained in Ω.

It follows immediately by the interior cone condition that the domain Ω contains balls with limited radius, for more details see [133].

The theorem below on local polynomial reproduction ensures the existence of an approximation scheme with local polynomial precision (see [133]).

The existence of an approximation scheme with local polynomial precision is exhibited by the following theorem (see [133]).

Theorem 1.4.11. *Suppose that $\Omega \subseteq \mathbb{R}^m$ is bounded and satisfies an interior cone condition, and let $s \in \mathbb{N}_0$. Then there exist constants h_0, c_1, $c_2 > 0$ such that for all $\mathcal{S}_n = \{\mathbf{x}_i, i = 1,2,\dots,n\} \subseteq \Omega$ with $h_{\mathcal{S}_n,\Omega} \leq h_0$ and all $\mathbf{x} \in \Omega$ there exist numbers $\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_n$ with*

(1) $\sum_{j=1}^n \tilde{u}_j(\mathbf{x})p(\mathbf{x}_j) = p(\mathbf{x})$ *for all polynomials* $p \in \Pi_s^m$;

(2) $\sum_{j=1}^n |\tilde{u}_j(\mathbf{x})| \leq c_1$;

(3) $\tilde{u}_j(\mathbf{x}) = 0$ *if* $\|\mathbf{x} - \mathbf{x}_j\|_2 \geq c_2 h_{\mathcal{S}_n,\Omega}$.

The attainment of polynomial precision is guaranteed by property (1), while property (3) demonstrates the locality. Property (2) provides a bound that controls the error growth, where $\sum_{j=1}^{n} |\tilde{u}_j(\mathbf{x})|$ represents the *Lebesgue constant* at $\mathbf{x}$.

The multivariate Taylor expansion of $\Phi(w, \cdot)$ centered at w can be expressed as

$$\Phi(w, z) = \sum_{|\beta|<2k} \frac{D_2^{\beta}\Phi(\mathbf{w}, \mathbf{w})}{\beta!} (z - w)^{\beta} + R(w, z),$$

with remainder $R(w, z)$ given by

$$R(w, z) = \sum_{|\beta|=2k} \frac{D_2^{\beta}\Phi(\mathbf{x}, \boldsymbol{\xi}_{w,z})}{\beta!} (z - w)^{\beta},$$

with $\boldsymbol{\xi}_{w,z}$ lying along the line segment connecting w to z.

Using the fill distance, the error estimate from Theorem 1.4.8 can now be refined.

The error estimate of the Theorem 1.4.8 can be refined by applying the notion of fill distance.

Theorem 1.4.12. *Let $\Omega \subseteq \mathbb{R}^m$ be bounded and satisfies an interior cone condition. Suppose that $\Phi \in C^{2k}(\Omega \times \Omega)$ is symmetric and strictly positive definite. Denote by I the interpolant to $f \in \mathcal{N}_{\Phi}(\Omega)$ on the set $\mathcal{S}_n$. Then there exist constants h_0, $C > 0$ (independent of $\mathbf{x}$, f and Φ) such that*

$$|f(\mathbf{x}) - I(\mathbf{x})| \leq C h_{\mathcal{S}_n,\Omega}^{k} \sqrt{C_{\Phi}(\mathbf{x})}\, \|f\|_{\mathcal{N}_{\Phi}(\Omega)},$$

provided $h_{\mathcal{S}_n,\Omega} \leq h_0$. Here

$$C_{\Phi}(\mathbf{x}) = \max_{|\beta|=2k,\, w,z \in \Omega \cap B(\mathbf{x}, c_2 h_{\mathcal{S}_n,\Omega})} \left| D_2^{\beta}\Phi(\mathbf{w}, \mathbf{z}) \right|$$

with $B(\mathbf{x}, c_2 h_{\mathcal{S}_n,\Omega})$ denoting the ball of radius $c_2 h_{\mathcal{S}_n,\Omega}$ centred at $\mathbf{x}$.

1.5 Stability and Trade-off Principles

1.5.1 RBF Interpolant Stability and Conditioning

In interpolation techniques, numerical stability is typically measured using the condition number. For radial basis function (RBF) interpolation, this involves the condition number of the interpolation matrix $\boldsymbol{A}$, where $A_{ij} = \Phi(\mathbf{x}_i - \mathbf{x}_j)$. The condition number is defined as

$$\text{cond}(\boldsymbol{A}) = \|\boldsymbol{A}\|_2 \|\boldsymbol{A}^{-1}\|_2 = \frac{\sigma_{\max}}{\sigma_{\min}},$$

where $\sigma_{\max}$ and $\sigma_{\min}$ are the largest and smallest singular values of $\boldsymbol{A}$, respectively. For positive definite matrices, it can be expressed as $\lambda_{\max}/\lambda_{\min}$, with $\lambda_{\max}$ and $\lambda_{\min}$ being the maximum and minimum eigenvalues.

Using Gershgorin's theorem [79], we have:

$$|\lambda_{\max} - A_{ii}| \leq \sum_{j=1, j\neq i}^{n} |A_{ij}|, \quad \text{for } i = 1,2,\ldots,n.$$

Thus,

$$\lambda_{\max} \leq n \max_{i,j=1,2,\ldots,n} |A_{ij}| = n \max_{\mathbf{x}_i,\mathbf{x}_j \in \mathcal{S}_n} |\Phi(\mathbf{x}_i - \mathbf{x}_j)|,$$

and given the strictly positive definiteness of Φ,

$$\lambda_{\max} \leq n\Phi(0).$$

For quasi-uniformly distributed points, $\lambda_{\max}$ grows at most as $h_{\mathcal{S}_n,\Omega}^{-m}$. The challenge lies in bounding $\lambda_{\min}$ or $\|\boldsymbol{A}^{-1}\|_2$. Researchers have explored this, leveraging Ball's results on eigenvalues of distance matrices using the Rayleigh quotient to find the smallest eigenvalue of a symmetric positive definite matrix as:

$$\lambda_{\min} = \min_{c\in\mathbb{R}^n\setminus 0} \frac{c^T \boldsymbol{A} c}{c^T c}.$$

Lower bounds on $\lambda_{\min}$ are expressed in terms of the separation distance between nodes:

$$q_{\mathcal{S}_n} = \frac{1}{2} \min_{i\neq j} \|\mathbf{x}_i - \mathbf{x}_j\|_2. \tag{1.11}$$

For more details, see [85, 86, 6, 7, 84, 107, 110] and the book [133]. As an example, for Gaussian functions $\Phi(\mathbf{x}) = e^{-\varepsilon^2\|\mathbf{x}\|^2}$, $\varepsilon > 0$ the explicit bound on $\lambda_{\min}$ is the following:

$$\lambda_{\min} \geq C_m(\sqrt{2}\varepsilon)^{-m} e^{-40.71m^2/(q_{\mathcal{S}_n}\varepsilon)^2} q_{\mathcal{S}_n}^{-m}.$$

As the separation distance $q_{\mathcal{S}_n}$ decreases, $\lambda_{\min}$ diminishes, leading to an increase in the condition number $\lambda_{\max}/\lambda_{\min}$. Thus, adding nodes to improve the accuracy results in causing higher ill-conditioning due to the decreasing separation distance.

Similarly, decreasing the shape parameter ε also increases the condition number, even if ε improves fit accuracy.

1.5.2 Trade-off Principles

Accuracy vs. Stability. The first *trade-off principle* (see, e.g., [106, 108]) highlights a conflict between theoretical accuracy and numerical stability. Infinitely smooth basis functions like Gaussians reduce errors exponentially as the fill distance decreases. However, this also reduces separation distance, increasing the condition number and causing numerical instability.

Reducing ε for accuracy also raises the condition number, as basis functions resemble constants, leading to matrix singularity.

Researchers seek an optimal shape parameter for balancing accuracy and stability (see, e.g., [60, 53, 19, 48, 20, 98]).

Accuracy and Stability vs. Problem Size. Some studies (see, e.g., [40, 49, 65, 94]) explore stability dependence on ε. They propose using complex Contour-Padé integration for accurate and stable computation as $\varepsilon \to 0$, but, since they are computationally expensive they are practical only for small datasets.

Accuracy vs. Efficiency. For compactly supported functions, there is a trade-off between stability and convergence. In stationary interpolation, where the support size of the basis functions is scaled with the fill distance, numerical stability is maintained, but convergence is limited.

In non-stationary interpolation with a fixed support size, decreasing the fill distance leads to denser and more ill-conditioned matrices. This approach achieves better accuracy but suffers from efficiency issues due to the increased ill-conditioning of the matrices.

1.6 Shape parameter dependent RBF

In the previous section, we outline how, for RBFs that depend on a shape parameter, the choice of it can influence the accuracy and the stability of the interpolation problem 1.1.1. Even though there exists RBF independent from shape parameter like polyharmonics, in this dissertation, we only consider RBFs that depend on a shape parameter $\varepsilon > 0$ that provide for $\mathbf{x}, \boldsymbol{z} \in \Omega$, the real symmetric strictly positive definite kernel

$$\kappa_\varepsilon(\mathbf{x}, \boldsymbol{z}) = \varphi(\varepsilon||\mathbf{x} - \boldsymbol{z}||_2) := \varphi(\varepsilon r).$$

The kernel-based interpolant I_f can be written as

$$I_f(\mathbf{x}) = \sum_{k=1}^{n} c_k \kappa_\varepsilon(\mathbf{x}, \mathbf{x}_k), \quad \mathbf{x} \in \Omega, \tag{1.12}$$

whose coefficients c_k are the solution of the linear system

$$\mathsf{K}_\varepsilon \boldsymbol{c} = \boldsymbol{f}, \tag{1.13}$$

where $\boldsymbol{c} = (c_1, \ldots, c_n)^\intercal$, $\boldsymbol{f} = (f_1, \ldots, f_n)^\intercal$, and $(\mathsf{K}_\varepsilon)_{ik} = \kappa_\varepsilon(\mathbf{x}_i, \mathbf{x}_k)$, $i, k = 1, \ldots, n$. In table 1.1 are reported some broadly used shape parameter dependent radial basis functions alongside their smoothness and abbreviations used in the dissertation (see [43, 133]).

1.7 Least square approximation

So far we have we have focused solely on interpolation. Nevertheless, sometimes it might be more appropriate to approximate the given data by a least squares RBF approximant. This is particularly true if the data includes noise or if there are so many data points that efficiency concerns necessitate approximating the data using fewer basis functions than there are data points. In this section, we consider a more general scenario where the function f is sampled at a set $\mathbf{X}_n$ of data points and the kernels are centred at $\tilde{\mathbf{X}}_s = \{\tilde{\mathbf{x}}_i, i = 1, \ldots, s\}$, which we will refer to as centres. A proper least squares approximation occurs when $s \leq n$ and the special case $s = n$ corresponds to RBF interpolation as described in Problem 1.1.1, where $\tilde{\mathbf{X}}_s = \mathbf{X}_n$.

RBF	Expression	Smoothness	Abbreviation
Gaussian	$\exp(-\varepsilon^2 r^2)$	C^∞	GA
Matérn C^0	$\exp(-\varepsilon r)$	C^0	M0
Matérn C^2	$\exp(-\varepsilon r)(1+\varepsilon r)$	C^2	M2
Matérn C^4	$\exp(-\varepsilon r)(3+3\varepsilon r+\varepsilon^2 r^2)$	C^4	M4
Matérn C^6	$\exp(-\varepsilon r)(15+15\varepsilon r+6\varepsilon^2 r^2+\varepsilon^3 r^3)$	C^6	M6
Wendland C^2	$\max(1-\varepsilon r,0)^4(4\varepsilon r+1)$	C^2	W2
Wendland C^4	$\max(1-\varepsilon r,0)^6(35\varepsilon^2 r^2+18\varepsilon r+3)$	C^4	W4
Wendland C^6	$\max(1-\varepsilon r,0)^8(32\varepsilon^3 r^3+25\varepsilon^2 r^2+8\varepsilon r+1)$	C^6	W6

Table 1.1: Popular Radial Basis Functions with their expression, abbreviation and smoothness.

In terms of least square the RBF approximant can be described as

$$\tilde{I}_f(\mathbf{x}) = \sum_{k=1}^{m} \tilde{c}_k \kappa_\varepsilon(\mathbf{x}, \tilde{\mathbf{x}}_k), \quad \mathbf{x} \in \Omega, \tag{1.14}$$

the coefficients $\tilde{c}_k$ being determined as the least squares solution of the linear system

$$\tilde{\mathsf{K}}_\varepsilon \tilde{\boldsymbol{c}} = \boldsymbol{f}, \tag{1.15}$$

obtained by minimizing the square of the 2-norm of the distance between the function f and the approximant $\tilde{I}_f$, $||\tilde{I}_f - f||_2^2$, where $(\tilde{\mathsf{K}}_\varepsilon)_{ik} = \kappa_\varepsilon(\mathbf{x}_i, \tilde{\mathbf{x}}_k)$, $i = 1, \ldots, n$, $k = 1, \ldots, s$, and $\boldsymbol{f} = (f_1, \ldots, f_n)^\intercal$. The uniqueness of the approximation problem is ensured if the collocation matrix $\tilde{\mathsf{K}}_\varepsilon$, with dimensions $n \times s$, has full rank. If the kernel is symmetric and strictly positive definite [43], the full rank property for the matrix $\tilde{\mathsf{K}}_\varepsilon$ is guaranteed if the centers $\tilde{\mathbf{X}}_s$ are a subset of the data locations $\mathbf{X}_n$.

Chapter 2

Partition of Unity Method

In order to contain the ill-conditioning and computational effort involved in solving large, dense linear systems, the Partition of Unity (PU) method is a useful tool for avoiding such problems. First introduced in the context of Partial Differential Equation (PDEs) to supersede the classical finite element method (FEM) approaches and shrink their computational expense [78, 5], it gained a foothold in the field of Approximation theory and was applied to different problems arising from various context like the financial [104, 100] or signal processing [27, 35].

2.1 Partition of Unity

The design on which the PU method is based is as simple as effective: given an open and bounded domain Ω, find a partition of *d subdomains or patches* Ω_j such that $\Omega \subseteq \cup_{j=1}^{d} \Omega_j$ with some overlap among them. The subdomains should collectively cover the entire domain, and additionally, there should be enough overlap to ensure that every point x within the domain Ω is also within the boundaries of at least one subdomain Ω_j. The covering can be obtained with subdomains of different shapes: circular, ellipsoidal, rhomboidal and so on but the most used and simpler is with circular shape. For the sake of displaying, in Figure 2.1 we consider, but it is not restrictive, a 2-dimensional circular PU covering of the unit square $\Omega = [0, 1]^2$. Notably, in the left frame the overlapping property is satisfied while the right frame is an example of covering that does not satisfy this property.

Together with these subdomains we consider a partition of unity weight functions w_j, a family of compactly supported, non-negative, continuous functions, linked to the subdomains Ω_j such that

i. $supp(X_j) \subseteq \Omega_j$,

ii. $\sum_{j=1}^{d} w_j(\mathbf{x}) = 1, \qquad \mathbf{x} \in \Omega$,

iii. $\|D^{\boldsymbol{\beta}} w_j\|_{L^\infty(\Omega_j)} \leq \frac{C_{\boldsymbol{\beta}}}{\delta_j^{|\boldsymbol{\beta}|}}, \forall \boldsymbol{\beta} \in \mathbb{N}^m : |\boldsymbol{\beta}| \leq k$, where $\delta_j = \sup_{\mathbf{x},\mathbf{y} \in \Omega_j} \|\mathbf{x} - \mathbf{y}\|$ and $C_{\boldsymbol{\beta}} > 0$ is a constant.

In accordance with [131], a family of non negative functions $\{w_j | j = 1, \ldots, d\}$ with $w_j \in C^k(\mathbb{R}^m)$ that satisfy i, ii and iii, is a k-stable partition of unity. A possible and

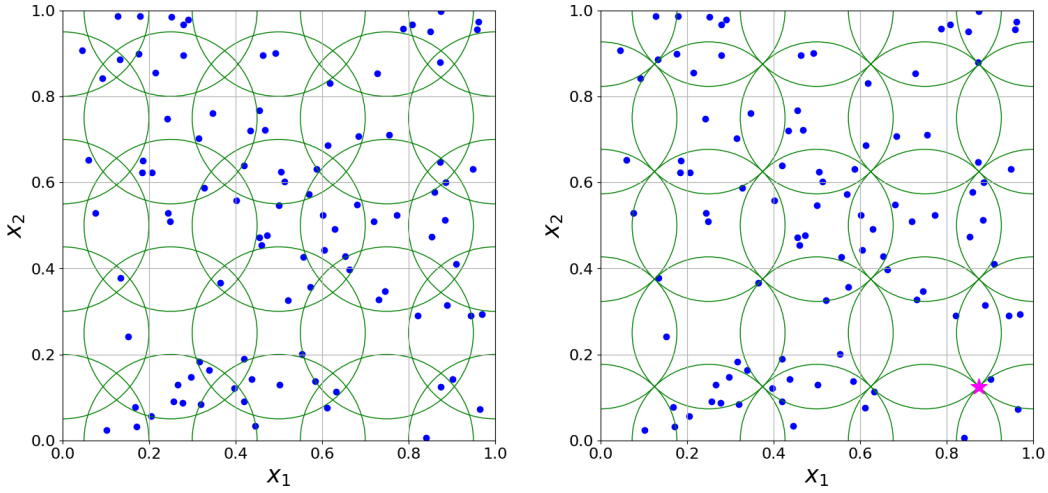

Figure 2.1: Examples of finite covering of the subdomain $\Omega = [0,1]^2$ On the left side a PU subdomain covering satisfies the overlapping condition. On the right, this condition is not satisfied and a magenta star highlights the critical point where no overlap occurs. The blue points are a set of random points in Ω and the green circles identify the PU patches.

most common choice for the non-negative weight functions $w_j \in C^k(\mathbb{R}^m)$ is given by the Shepard's weights [105]:

$$w_j(\mathbf{x}) = \frac{\bar{w}_j(\mathbf{x})}{\sum_{k=1}^{d} \bar{w}_k(\mathbf{x})}, \qquad j = 1, \dots, d, \tag{2.1}$$

where $\bar{w}_j(\mathbf{x})$ are compactly supported functions with support on Ω_j such as Wendland's functions [133]. A further requirement for the applicability of the Partition of unity method [131] is the regularity of the subdomains, formally exposed in the following definition:

Definition 2.1.1. Given $\Omega \subseteq \mathbb{R}^m$ bounded and $X_n = \{\mathbf{x}_i, i = 1, \dots, n\} \subseteq \Omega$. An open and bounded covering $\{\Omega_j : j = 1, \dots, d\}$ is regular for (Ω, X_n) if the following properties are fulfilled:

i. for each $x \in \Omega$, the number of subdomains Ω_j, with $x \in \Omega_j$, is bounded by a global constant C,

ii. there exists a constant $C_r > 0$ and an angle $\theta \in (0, \frac{\pi}{2})$ such that every subdomain Ω_j satisfies an interior cone condition with angle θ and radius $r = C_r h_{X_n,\Omega}$,

iii. The local fill distances $h_{X_{n_j}}$ are uniformly bounded by the global fill distance h_{X_n}, where $X_{n_j} = X_n \cap \Omega_j$.

Once we have the regular open bounded covering and a k-stable family of compactly supported, non-negative, continuous function, the global interpolant can be written as a weighted sum of d local interpolants I_f^j defined on the subdomains Ω_j of the form (1.12):

$$I_f(\mathbf{x}) = \sum_{j=1}^{d} I_f^j(\mathbf{x}) w_j(\mathbf{x}), \qquad \mathbf{x} \in \Omega. \tag{2.2}$$

Remark 2.1.2. It is important to note that, according to Definition 2.1.1, the number of subdomains is proportional to the amount of data. Moreover, property (i) ensures that the sum in (2.2) involves at most C terms. The independence between C and n is crucial to maintain convergence rates. Additionally, it is essential to evaluate only a constant number of local approximants to achieve efficient computation of the global interpolant.

2.2 RBF-PUM interpolant error bound

Let $C^k_\nu(\mathbb{R})^m$ the space of all functions $f \in C^k$ whose derivatives of order $|\beta| = k$ satisfy $D^\beta f(\mathbf{x}) = \mathcal{O}(\|\mathbf{x}\|_2^\nu)$ for $\|\mathbf{x}\|_2 \longrightarrow 0$, In order to formulate an error bound, the following convergence result is considered [43, 133]:

Theorem 2.2.1. *Let $\Omega \subseteq \mathbb{R}^m$ be an opened and bounded and suppose that $X_n = \{\mathbf{x}_i, i = 1, \ldots, n\} \subseteq \Omega$. Let $\varphi \in C^k_\nu(\mathbb{R}^m)$ be a strictly conditionally positive definite function of order s. Let $\{\Omega_j\}_{j=1}^d$ be a regular covering for (Ω, X_n) and $\{w_j\}_{j=1}^d$ be k-stable for $\{\Omega_j\}_{j=1}^d$. Then the error between $f \in \mathcal{N}_\varphi(\Omega)$, where $\mathcal{N}_\varphi$ is the native space of φ, and its PU interpolant (2.2) can be bounded by*

$$|D^\beta f(\mathbf{x}) - D^\beta I(\mathbf{x})| \leq C' h_{X_n}^{(k-\nu)/2-|\beta|} |f|_{\mathcal{N}_\varphi(\Omega)},$$

for all $\mathbf{x} \in \Omega$ and all $|\beta \leq \frac{k}{2}$.

From the error bounds in section 1.4.3 and the Theorem 2.2.1, it can be seen that the *PU* interpolant preserves the local approximation order for the global fit. Therefore, large *RBF* interpolant can be computed by solving interpolation problems on small portions of data and merging together globally with the partition of unity $\{w_j\}_{j=1}^d$.

2.3 The partitionunity package

An earlier version of the partition of unity implementation relying on kd-tree partitioning data structures [43] exists but it utilizes files with a dll extension (commonly referred to as mex files), which are incompatible with recent MATLAB versions. To address this issue, we present a new implementation of the PU scheme using what we term integer-based routines. While the underlying theory of these data partition routines is not new and can be viewed as a multivariate extension of the algorithms presented in [21], we provide a comprehensive discussion of their implementation here. The remainder of this chapter revisits topics covered in the published work [29], co-authored by the author of this thesis.

2.3.1 MATLAB PU method implementation

To provide an explanation of how the PU method works we provide code examples in MATLAB language, including the script 2.1 and the function `PU.m` in the function 2.2. The covering for the PU consists of overlapping balls with fixed radii, centred at mesh data points $P = \{\tilde{\boldsymbol{x}}_k,\ k = 1, \ldots, d\}$; see the code line 16 of the function 2.2, where `m_d` denotes the number of subdomains in one direction[1]. Assuming a nearly uniform distribution of

[1] The function `MakeSDGrid.m` is provided by [43].

nodes, and following [43], a suitable number of subdomains for the PU on Ω is d if

$$\frac{n}{d} \approx 2^m.$$

To satisfy the covering property, we choose the radius δ such that

$$\delta \geq \frac{1}{d^{1/m}}.$$

The number of patches in each dimension and the radius are fixed. We chose $\lfloor n^{1/m}/2 \rfloor$ for the number of patches in line 8 of the script 2.1 and $\delta = \sqrt{2}/d^{1/m}$ in line 17 of the function 2.2.

At this point a local interpolation problem is solved on each subdomain and the PU method composes these local approximants using some weights functions producing the global interpolant.

The weights used in the code are the Shepard's weights previously introduced: at line 9 of the script 2.1 we define `w` that is the Wendland C^2 function [133] while we construct the weights at line 19 of function 2.2. Once we choose the partition of unity $\{w_j\}_{j=1}^d$, the global interpolant is formed by the weighted sum of d local approximants I_f^j, i.e.

$$I_f(\boldsymbol{x}) = \sum_{j=1}^{d} I_f^j(\boldsymbol{x})\, w_j(\boldsymbol{x}) = \sum_{j=1}^{d} \left(\sum_{k=1}^{n_j} c_k^j \kappa(\boldsymbol{x}, \boldsymbol{x}_k^j) \right) w_j(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega,$$

where $X_j = X_n \cap \Omega_j$, $n_j = |X_j|$ and $\boldsymbol{x}_k^j \in X_j$, for $k = 1, \ldots, n_j$. The functions `IntegerBasedContainingQuery.m` (function 2.3), `IntegerBasedStructure.m` (function 2.4), `IntegerBasedNeighbourhood.m` (function 2.5) and `IntegerBasedNeighbourhood` (function 2.6), pertain the organization of points in the subdomains and will be discussed in the following section. After the distribution of data among the subdomain, the coefficients $\{c_k^j\}_{k=1}^{n_j}$ must be determined by imposing the n_j local interpolation conditions:

$$I_f^j(\boldsymbol{x}_i^j) = f_i^j, \quad i = 1, \ldots, n_j,$$

that are equivalent to solve d linear systems of the form

$$\mathsf{K}_j \boldsymbol{c}_j = \boldsymbol{f}_j,$$

where $\boldsymbol{c}_j = (c_1^j, \ldots, c_{n_j}^j)^T$, $\boldsymbol{f}_j = (f_1^j, \ldots, f_{n_j}^j)^T$ and $\mathsf{K}_j \in \mathbb{R}^{n_j \times n_j}$ is[2]

$$\mathsf{K}_j = \begin{pmatrix} \kappa(\boldsymbol{x}_1^j, \boldsymbol{x}_1^j) & \cdots & \kappa(\boldsymbol{x}_1^j, \boldsymbol{x}_{n_j}^j) \\ \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{x}_{n_j}^j, \boldsymbol{x}_1^j) & \cdots & \kappa(\boldsymbol{x}_{n_j}^j, \boldsymbol{x}_{n_j}^j) \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} \phi\left(||\boldsymbol{x}_1^j - \boldsymbol{x}_1^j||_2\right) & \cdots & \phi\left(||\boldsymbol{x}_1^j - \boldsymbol{x}_{n_j}^j||_2\right) \\ \vdots & \ddots & \vdots \\ \phi\left(||\boldsymbol{x}_{n_j}^j - \boldsymbol{x}_1^j||_2\right) & \cdots & \phi\left(||\boldsymbol{x}_{n_j}^j - \boldsymbol{x}_{n_j}^j||_2\right) \end{pmatrix}}_{\texttt{K\_j=phi(ep,DM\_data)}}.$$

[2]The function `DistanceMatrix.m` is provided by [43].

The corresponding systems solution is reported in lines 32–33 of the function 2.2. The aim of the global PU interpolant is to estimate the values of the unknown function on a set of evaluation data. In particular, we consider a grid of points as test or evaluation data $E = \{\bar{\boldsymbol{x}}_\ell,\ \ell = 1, \dots, s\}$. For each subdomain, after the determination of the coefficients $\boldsymbol{c}^j$ by solving the local interpolation system on Ω_j, the local interpolant is evaluated on the set $E_j = E \cap \Omega_j$ with $s_j = |E_j|$ as in function 2.2, line 34–35.

$$\begin{pmatrix} P_f^j\left(\bar{\boldsymbol{x}}_1^j\right) \\ \vdots \\ P_f^j\left(\bar{\boldsymbol{x}}_{s_j}^j\right) \end{pmatrix} = \begin{pmatrix} \phi\left(||\bar{\boldsymbol{x}}_1^j - \boldsymbol{x}_1^j||_2\right) & \cdots & \phi\left(||\bar{\boldsymbol{x}}_1^j - \boldsymbol{x}_{n_j}^j||_2\right) \\ \vdots & \ddots & \vdots \\ \phi\left(||\bar{\boldsymbol{x}}_{s_j}^j - \boldsymbol{x}_1^j||_2\right) & \cdots & \phi\left(||\bar{\boldsymbol{x}}_{s_j}^j - \boldsymbol{x}_{n_j}^j||_2\right) \end{pmatrix} \begin{pmatrix} c_1^j \\ \vdots \\ c_{n_j}^j \end{pmatrix}.$$

The final evaluation on the whole grid E is carried out by summing up all the local contributes weighted by the matrix $W \in \mathbb{R}^{s \times d}$ of compactly supported weights as in line 36 of the function 2.2. To give an example, we report the surface interpolating the Franke's function (see Figure 2.2) at the 4225 data, as the script 2.1 does. As error indicator, we take the Maximum Absolute Error (MAE) defined as:

$$\text{MAE} := \max_{i=1,\dots,s} |P_f(\bar{x}_i) - f(\bar{x}_i)|.$$

In the considered example, we obtain $\text{MAE} = 6.67\text{E} - 04$.

```
1                          % Example 1
2  m = 2; n = [65].^m; % Define the space dimension and the number of data
3  p = haltonset(m); x = net(p,n); % Define the interpolation data (Halton
       points)
4  phi = @(epsilon,r) (1+epsilon*r).*exp(-epsilon*r); epsilon = 1; % Define
       the kernel
5  franke = @(x1,x2) 0.75 * exp(-(9*x1-2).^2/4 - (9*x2-2).^2/4)+ 0.75 * exp
       (-(9*x1+1).^2/49 - ...
6      (9*x2+1)./10)+0.5 * exp(-(9*x1-7).^2/4 - (9*x2-3).^2/4)-0.2 * exp(-(9*
       x1-4).^2 - (9*x2-7).^2);
7  f = franke(x(:,1),x(:,2)); % Define the test function and the function
       values
8  d_m = floor(n^(1/m)/2); s_m = 60; % Define the number of patches and
       evaluation data in one direction
9  w = @(supp,r) (max(1-(supp*r),0).^4).*(4*(supp*r)+1); % Define the PU
       weights (Wendland C^2)
10 bar_x = MakeSDGrid(m,s_m); % Create s_m^d equally spaced test data
11 Pf = PU(m,x,bar_x,d_m,phi,w,f,epsilon); % Compute the PU interpolant
```

script 2.1: Template script for computing the PU interpolant.

```
1  function [Pf] = PU(m,x,bar_x,d_m,phi,w,f,epsilon)
2
3  Goal: script that performs partition of unity
4
5  Inputs:  m: space dimension
```

```
6           x: nXd matrix representing a set of n interpolation data
7           bar_x: sXd matrix representing a set of s evaluation data
8           d_m: number of PU subdomains in one direction
9           phi: radial basis function
10          w: weight function
11          f: the function values
12          epsilon: the shape parameter
13
14  Outputs:  Pf: sXd matrix representing the PU fit
15
16  tilde_x = MakeSDGrid(m,d_m); % Create d_m^m equally spaced PU centres
17  delta = sqrt(2)/d_m;  supp = 1/delta; % Define the PU radius and the
        parameter for the weight functions
18  m = size(tilde_x,1); s = size(bar_x,1); Pf = zeros(s,1);  % Initialize and
        compute the Shepard matrix
19  DM_eval = DistanceMatrix(bar_x,tilde_x); W = w(supp,DM_eval); W = spdiags
        (1./(W*ones(m,1)),0,s,s)*W;
20  q = ceil(1./delta); % Parameter for the integer-based partitioning
        structure
21  % Build the partitioning structure for interpolation and evaluation data
22  X_block = IntegerBasedStructure(x,q,delta,m); bar_X_block =
        IntegerBasedStructure(bar_x,q,delta,m);
23  for j = 1:m % Loop over subdomains
24      k = IntegerBasedContainingQuery(tilde_x(j,:),q,delta,m); % Find the box
         with the j-th PU centre
25      % Find the interpolation data located in the j-th subdomain
26      [X_NeigBlock, idx_X_NeigBlock] = IntegerBasedNeighbourhood(x,X_block,k,
        q,m);
27      n_j = IntegerBasedRangeSearch(tilde_x(j,:),delta,X_NeigBlock,
        idx_X_NeigBlock);
28      % Find the evaluation data located in the j-th subdomain
29      [bar_X_NeigBlock, idx_bar_X_NeigBlock] = IntegerBasedNeighbourhood(
        bar_x,bar_X_block,k,q,m);
30      s_j = IntegerBasedRangeSearch(tilde_x(j,:),delta,bar_X_NeigBlock,
        idx_bar_X_NeigBlock);
31      if  (~isempty(s_j)) &&  (~isempty(n_j))
32          DM_data = DistanceMatrix(x(n_j,:),x(n_j,:)); K_j = phi(epsilon,
        DM_data); % Interpolation matrix
33          c_j = K_j\f(n_j);  % Compute the interpolation coefficients
34          DM_eval = DistanceMatrix(bar_x(s_j,:),x(n_j,:)); % Compute the
        evaluation matrix
35          K_eval = phi (epsilon, DM_eval); P_fj = K_eval*c_j; % Compute the
        local fit
36          Pf(s_j) = Pf(s_j) + P_fj.*W(s_j,j); % Accumulate the global fit
37      end
```

```
38 end
```

function 2.2: PU method MATLAB implementation.

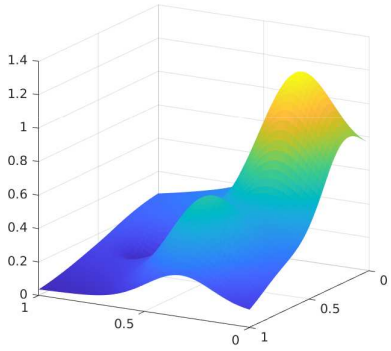

Figure 2.2: Representation of the PU interpolant surface produced by the script 2.1.

2.3.2 Data partitioning structure

In this subsection we deal with the organizing procedure to distribute the data among the different patches. Firstly introduced in [34] the integer based routines can be considered as a multidimensional improvement of the cross-strip algorithms investigated in [21] and later deepen in [25]. A supporting structure used to distribute the points into the different patches are the blocks structure that consists in dividing Ω into several blocks (also called boxes). The number of blocks q along one dimention of $[0,1]^m$ depends on δ, the radius of PU subdomains, and it is equal to

$$q = \left\lceil \frac{1}{\delta} \right\rceil. \tag{2.3}$$

We evaluate q in line 20 of the function 2.2.

The blocks are numbered from 1 to q^m, starting from the subspace of dimension $m-1$, obtained projecting along the first coordinate and thus parallel to the remaining ones. Then to organize the points we use the so-called integer-based routines discussed in what follows.

The first query we have to answer is: given a PU centre find the index of the block it belongs to. We observe that, given a PU centre $\bar{\boldsymbol{x}}_j$, the index of the k-th block containing the subdomain centre is given by (see line 13 of the Matlab function 2.3)

$$k = \sum_{\ell=1}^{m-1} (k_\ell - 1)\, q^{m-\ell} + k_m. \tag{2.4}$$

In order to determine the values of the indices k_ℓ, $\ell = 1, \ldots, m$ that appear in (2.4), an integer-based procedure is applied. It lies in rounding off the ratio between the l-component

of a PU centre and the radius. For a given PU centre $\bar{\boldsymbol{x}}_j = (\bar{x}_{j1}, \ldots, \bar{x}_{jd})$, k_ℓ is given by:

$$k_\ell = \left\lceil \frac{\bar{x}_{j\ell}}{\delta} \right\rceil.$$

```
function [k] = IntegerBasedContainingQuery(tilde_x,q,delta,m)
%
% Goal: script that given a subdomain centre returns the index of the
    square block containing the
%       subdomain centre
%
% Inputs: tilde_x: subdomain centre
%         q: number of blocks in one direction
%         delta: radius of the PU subdomains
%         m: space dimension
%
% Outputs: k: the index of the block containing the subdomain centre
%
k_l = ceil(tilde_x/delta); l = 1:m-1; k_l(k_l == 0) = 1; k = sum((k_l(l)-1)
    *q.^(m-l)) + k_l(end);
```

function 2.3: Implementation of the integer-based containing query procedure.

As for the PU centres, the same rounding-off approach is used to settle the scattered data and the evaluation points into the boxes. The data structure we build allows us to save for each subdomain all the indices of the data that fall within and this procedure is performed by the function given in the function 2.4.

```
function [X_block] = IntegerBasedStructure(x,q,delta,m)
%
% Goal: find the data sites located in each of the q^m blocks
%
% Inputs: x: nXd matrix representing a set of data
%         q: number of blocks in one direction
%         delta: radius of PU subdomains
%         m: space dimension
%
% Outputs: X_block: multiarray containing the indices of the data points
    located in k-th block
%
n = size(x,1); X_block = cell(q^m,1); k = 1:m-1; % Initialize
for i = 1:n % Find the indices of the data points located in k-th block
    idx = ceil(x(i,:)/delta); idx(idx == 0) = 1; index = sum((idx(k)-1)*q
    .^(m-k)) + idx(end);
    X_block{index} = [X_block{index}; i];
end
```

function 2.4: Implementation of the integer-based data structure.

Once the containing query routine is implemented and we retrieve the indices of data in each box we need to determine the interpolation and evaluation data in each subdomain. In other words, suppose to have a set of interpolation data X, a set of evaluation data E and a subdomain Ω_j, we want to determine all the interpolation data and all the evaluation data in that subdomain:

- $\boldsymbol{x}_i \in X_j, \quad i = 1, \dots, n_j \quad$ (interpolation data),
- $\bar{\boldsymbol{x}}_i \in E_j, \quad i = 1, \dots, s_j \quad$ (evaluation data).

If we set q as in (2.3), for each subdomain centre $\tilde{\boldsymbol{x}}_j$ belonging to the block k, the neighbour points must be searched in the block k and in its $3^m - 1$ neighbouring blocks determined by the function 2.5.

```
function [X_NeigBlock, idx_X_NeigBlock] = IntegerBasedNeighbourhood(x,
    X_block,k,q,m)
%
% Goal: script that finds the neighbouring blocks
%
% Inputs:  x: nXd matrix representing a set of n data sites
%          X_block: the integer-based data structure
%          k: the k-th block cointaining the subdomain centre
%          q: number of blocks in one direction
%          m: space dimension
%
% Outputs:  X_NeigBlock, dx_X_NeigBlock: points (and indices) lying in the
    k-th neighbourhood
%
neigh = []; l = m-1; index = k; % Initialize
while l  > 0 % Find neighbouring blocks
     neigh = [k+q.^l,k-q.^l];
    if l - 1 > 0
        neigh = [neigh,neigh+q.^(l-1),neigh-q.^(l-1)];
    end
    l = l - 1;
end
k2 = 1; neighplus = []; neighminus = []; % Initialize
for i = 1:length(neigh)
    neighplus(k2) = neigh(i) + 1; neighminus(k2) = neigh(i) - 1; k2 = k2 +
    1;
end
neigh = [neigh,k+1,k-1,neighplus,neighminus]; % Reduce the number of blocks
     for border blocks
j = find(neigh > 0 & neigh <= q^m); index = [index; neigh(j)']; X_NeigBlock
     = []; idx_X_NeigBlock = [];
for p = 1:length(index)
    X_NeigBlock = [X_NeigBlock;x(X_block{index(p)},:)];
    idx_X_NeigBlock = [idx_X_NeigBlock;X_block{index(p)}];
```

```
end
```

function 2.5: Implementation of the integer-based neighbourhood routine.

Notably, the block-based partitioning structure allows the exploration of at most $3^m - 1$ boxes during the searching process; if a boundary box is considered, the number of neighbouring blocks to explore is reduced. Lastly, by means of the so-called range search procedure, speeded up by a sorting routine (line 18 of the function 2.6), the points that lie in the subdomain j are found.

```
function [n_j] = IntegerBasedRangeSearch(tilde_x,delta,X_NeigBlock,
    idx_X_NeigBlock)
%
% Goal: find the data sites located in a given subdomain and the distances
    between the
%       subdomain centre and data sites
%
% Inputs:  tilde_x: subdomain centre; delta: radius of PU subdomains
%          X_NeigBlock: nXd matrix representing a set of n points
%          idx_X_NeigBlock: vector containing the indices of the data
    points located in the k-th block
%                           and in the neighbouring blocks
%
% Outputs: n_j:  vector containing the indices of the data points located
    in a given PU subdomain
%
N = size(X_NeigBlock,1); n_j = []; % Initialize
for i = 1:N % Compute distances between the data sites and the centre
    dist1(i) = norm(tilde_x - X_NeigBlock(i,:));
end
if N > 0 % Use a sort procedure to order distances
    [sort_dist,IX] = sort(dist1); N1 = size(sort_dist,2); j1 = 1; j2 = 1; %
    Initialize
    while (j2 <= N1) && (sort_dist(j2) <= delta) % Find the data sites
    located in the given subdomain
        n_j(j1) = idx_X_NeigBlock(IX(j2)); j1 = j1 + 1; j2 = j2 + 1;
    end
end
```

function 2.6: Implementation of the range search routine.

2.4 Python PU method implementation

Due to the increasing interest in Python packages within the kernel community for machine learning, we have also developed a Python implementation of the PU scheme. This implementation closely follows the Matlab version, utilizing some of Python's native structures. We have aimed to keep the variable and function names consistent with the Matlab

version to simplify code comparison. Additionally, we utilize libraries such as NumPy [36] and SciPy [126]. For more information about the method, refer to Section 2.3.1. The function 2.7 mirrors the functionality of the MATLAB counterpart function 2.2, with the main difference being the use of a *list comprehension* to generate the grid of equally spaced centres.

```
def PU(x, f, bar_x, d_m, w,  phi, epsilon):
    """
    Goal: build the partition of unity interpolant and approximate
          the values on a given set of points

    Inputs: x: nxd numpy array  representing a set of n data sites
            f: the function values
            bar_x: sxd numpy array  representing a set of s evaluation data
            d_m: number of PU subdomains in one direction
            w: weight function
            phi: radial basis function
            epsilon: the shape parameter

    Outputs: Pf: sXd numpy array representing the PU fit
    """

    m = x.shape[0]
    s, _ = bar_x.shape
    Pf = np.zeros(s)

    # Create d_m^m equally spaced PU centres
    tilde_x = np.array([i for i in itertools.product(np.linspace(0, 1, int(
    d_m)), repeat=m)])
    # Define the PU radius and the parameter for the weight functions
    radius_par = 2 ** (1 / 2)
    delta = radius_par / d_m
    supp = 1 / delta

    # Parameter for the integer-based partitioning structure
    q = np.ceil(1 / delta)

    # Build the partitioning structure for interpolation and evaluation
    data
    X_block = integer_based_structure(x, q, delta, m)
    bar_X_block = integer_based_structure(bar_x, q, delta, m)

    # Initialize and compute the Shepard matrix
    sem = w(supp, distance_matrix(bar_x, tilde_x))
    sem = spdiags(1/(sem@np.ones(int(d_m)**m)), 0, s, s)@sem

```

```
    # Loop over subdomains
    for j, center in enumerate(tilde_x):

        # Find the box with the j-th PU centre
        k = integer_based_containing_query(center, q, delta, m)
        # Find the interpolation data located in the j-th subdomain
        X_neig_block = integer_based_neighbourhood(k, q, m)
        n_j = integer_based_range_search(center, delta, x, X_block,
    X_neig_block)

        if n_j.size != 0:
            # Interpolation matrix
            c_j = np.linalg.solve(phi(epsilon, distance_matrix(x[n_j, :], x
    [n_j, :])), f[n_j])
            bar_X_neig_block = integer_based_neighbourhood(k, q, m)
            s_j = integer_based_range_search(center, delta, bar_x,
    bar_X_block, bar_X_neig_block)

            if s_j.size != 0:
                # Compute the local fit
                p_fj = np.dot(phi(epsilon, distance_matrix(bar_x[s_j, :], x
    [n_j, :])), c_j)
                # Accumulate the global fit
                Pf[s_j] += p_fj * np.array(sem[s_j, j])
    return Pf
```

Python function 2.7: Implementation of the PU method.

The function 2.8 is the Python translation of the function 2.5. Given a subdomain centre it applies the integer-based procedure to determine the block it belong to.

```
def integer_based_containing_query(tilde_x, q, delta, m):
    """
    Goal:   given a subdomain centre returns the index of the
            square block containing the subdomain centre

    Inputs: tilde_x: subdomain centre
            q: number of blocks in one dimension
            delta: radius of PU subdomains
            m: space dimension

    Outputs: k: the block containing the subdomain centre
    """
    k = np.ceil(tilde_x/delta)
    k[k == 0] = 1
    k = np.sum((k[:-1] - 1) * q ** np.arange(m - 1, 0, -1)) + k[-1]
```

```
    return k
```

Python function 2.8: Implementation of the integer-based containing query procedure.

To place the data in the blocks, in function 2.9 it is performed the same step is made to find the block a given subdomain centre belongs. The aim is to create an array of indices of the contained points for each block. The use of a dictionary structure is the main difference with the correspondent MATLAB function 2.4.

```
def integer_based_structure(x, q, delta, m):
    """
    Goal: find the data sites located in each of the q^m blocks

    Inputs: x: nxd numpy array  representing a set of n data sites
            q: number of blocks in one dimension
            delta: radius of PU subdomains
            m: space dimension

    Outputs: X_block: dictionary {key: values} key represent the index
             of the block  and values is a ndarray that contain indexes
             of points located in k-th block
    """

    X_block = {}
    n, _ = x.shape
    for ind in range(n):
        index = np.ceil(x[ind]/delta)
        index[index == 0] = 1
        index = int(np.sum((index[:-1] - 1)*q**np.arange(m-1, 0, -1)) +
    index[-1])
        if index in X_block.keys():
            X_block[index] = np.append(X_block[index], [ind], axis=0)
        else:
            X_block[index] = np.array([ind])
    return X_block
```

Python function 2.9: Implementation of the integer-based data structure.

The function 2.10, similar to its parallel Matlab version (function 2.5), identifies the neighboring blocks and their indices for a given block. This neighborhood construction procedure is particularly useful because it allows the searching procedure to be applied only to the neighborhood, rather than the entire dataset, resulting in significant computational time savings.

```
def integer_based_neighbourhood(k, q, m):
    """
    Goal: given a block finds neighbouring blocks
```

```
    Inputs:  k: index of the block
             q: number of blocks in one direction
             m: space dimension

    Outputs: X_NeigBlock: indices of the neighbouring blocks
    """

    neigh = np.array([])
    ld = m - 1
    while ld > 0:
        neigh = np.append(neigh, [k + q ** ld, k - q ** ld])
        if ld - 1 > 0:
            neigh = np.append(neigh, np.array([neigh + q ** (ld - 1), neigh
    - q ** (ld - 1)]))
        ld -= 1

    neigh = np.append(neigh, k)
    neigh = np.append(neigh, [neigh + 1, neigh - 1])
    X_neig_block = neigh[np.logical_and(neigh > 0, neigh <= q ** m)]
    return X_neig_block
```

Python function 2.10: Implementation of the integer-based neighbourhood routine.

The final code, function 2.11, implements the searching process that results in an array containing the points belonging to a subdomain. Notably, given the subdomain center and the block-based structures built by calling functions 2.9 and 2.10, the function 2.11 returns an array containing all the points in the specified subdomain.

```
def integer_based_range_search(tilde_x, delta, x,  X_block,
    idx_X_neigh_block):
    """
    Goal:   find the data sites located in a given subdomain

    Inputs: tilde_x: subdomain centre
            delta: radius of PU subdomain
            x: nxd numpy array  representing a set of n data sites
            idx_X_neigh_block:  dictionary {key: value} key represent
                                the index of the block  and value
                                is a list that contain indexes of
                                points located in k-th block  and in
                                the neighbouring blocks

    Outputs: n_j:  list of the indexes of the points belonging to a
                   given subdomain
    """

```

```
    if idx_X_neigh_block.size != 0:
        n_j = []
        for key in idx_X_neigh_block:
            try:
                for ind in X_block[key]:
                    if np.linalg.norm(tilde_x - x[ind]) <= delta:
                        n_j.append(ind)
            except KeyError:
                pass
        n_j = np.array(n_j)
    else:
        n_j = np.arange(x.shape[0])
    return n_j
```

Python function 2.11: Implementation of the range search routine.

2.5 Numerical examples

All the tests reported in this section are performed using Python 3.9.12 on a 1.2 GHz Quad-Core Intel Core i7 processor, 16 GB 3733 MHz LPDDR4X RAM MacBook Air (2020). To reproduce the results a Python version 3.8 or newer is required for the installation of the *partitionunity* package from the Python Package Index. The command for the installation of the packages depends on the OS. Unix/macOS:

```
python3 -m pip install partitionunity
```

Windows:

```
py -m pip install partitionunity
```

The import of the packages can be done as shown in the script 2.12 with the command

```
import partitionunity
```

we use the alias *pu* to avoid repeating the full name of the package. Figure 2.3 displays the Mean Absolute Error (MAE) and CPU times for the calculation of the Partition of Unity (PU) interpolant. To demonstrate the effectiveness of the PU implementation, we selected $n = \lfloor 7^{p/2} \rfloor^2$ for $p = 2, 3, 4, 5, 6$, using equally spaced interpolation data points, sample values from the Franke function, and the Matérn C^2 kernel as the Radial Basis Function (RBF). Readers can reproduce these results by executing the script provided in 2.12, which generates the plots shown in Figure 2.3

```
import partitionunity as pu
import numpy as np
import time
import matplotlib.pyplot as plt

# Define the test function
def franke(X):
    return 0.75 * np.exp(-(9*X[:, 0]-2)**2/4 - (9*X[:, 1]-2)**2/4) + \
```

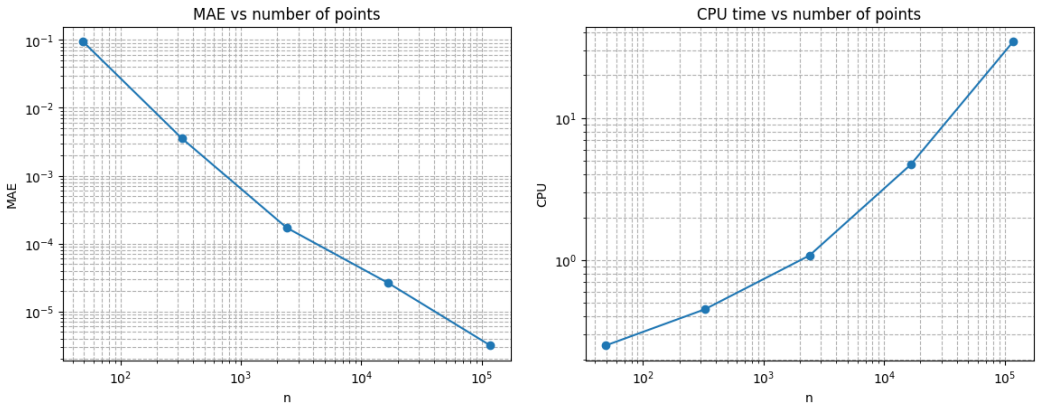

Figure 2.3: Log-Log representation of the maximum absolute error and execution time as the number of interpolation points increases.

```
            0.75 * np.exp(-(9*X[:, 0]+1)**2/49 - (9*X[:, 1]+1) / 10) + 0.5 *
    \
            np.exp(-(9*X[:, 0]-7)**2/4 - (9*X[:, 1]-3)**2/4)-0.2 * \
            np.exp(-(9*X[:, 0]-4)**2 - (9*X[:, 1]-7)**2)


# Define the space dimension and the number of interpolation data
m = 2
p = np.append(2, np.arange(2, 7))
n = np.floor(np.sqrt(7**p)).astype(int)


# Define the kernel and parameter
def Phi(eps, r):
    return (1 + eps * r) * np.exp(-eps * r)


epsilon = 1

# Define the weights for PU
def weight(e, r):
    return np.multiply(np.power(np.fmax(1-(e*r), 0*(e*r)), 4), (4*(e*r)+1))


# Define s_m^m equally spaced test data
s_m = 60
bar_x = np.array(np.meshgrid(np.linspace(0, 1, s_m), np.linspace(0, 1, s_m)
    )).T.reshape(-1, m)
t = []
```

```
MAE = []
for i in range(len(p)):

    # Define the interpolation data
    x = np.array(np.meshgrid(np.linspace(0, 1, n[i]), np.linspace(0, 1, n[i
    ]))).T.reshape(-1, m)

    # Define the function values
    y = franke(x)

    d_m = np.floor((n[i] ** 2) ** (1 / m) / 2)
    tm = time.time()

    # Compute the PU interpolant
    Pf = pu.PU(x, y, bar_x, d_m, weight, Phi, epsilon)

    t.append(time.time() - tm)
    MAE.append(np.max(abs(franke(bar_x) - Pf)))

# Display the results
plt.loglog(n[1:]**2, MAE[1:], 'o-', )
plt.title("MAE vs number of points")
plt.xlabel("n")
plt.ylabel("MAE")
plt.grid(True, which="both", linestyle='--')
plt.show()

plt.loglog(n[1:]**2, t[1:], 'o-')
plt.title("CPU time vs number of points")
plt.xlabel("n")
plt.ylabel("CPU(s)")
plt.grid(True, which="both", linestyle='--')
plt.show()
```

Python script 2.12: Usage example of the Python package.

As shown in Figure 2.3 the interpolation on more than 100000 data take place in a short reasonable time confirming the numerical efficiency of the PU algorithm. Further investigation could explore other applications of the PU method beyond interpolation, such as regression and support vector machines [74], or applying the PU approach to local fits provided by the Scikit-Learn package [91] or the MATLAB Statistics and Machine Learning Toolbox.

Chapter 3

Optimizers

This chapter introduces the optimizers utilized throughout the dissertation. In kernel-based interpolation, determining the shape parameter is crucial due to its significant impact on the interpolant's accuracy. The search for an optimal shape parameter remains a key research area, and the radius parameter in the Partition of Unity framework (see Chapter 3) is also subject to optimization. We present a modified variant of Leave-One-Out Cross-Validation (LOOCV), a widely used method for parameter search in RBF approximation. Additionally, we explore Bayesian Optimization (BO), an advanced statistical technique that guides parameter search using a probabilistic model.

3.1 Leave-One-Out Cross-Validation

A widely used technique for determining the optimal shape parameter ε in Radial Basis Function (RBF) interpolation, based on the given data set (X, F), is the Leave-One-Out Cross-Validation (LOOCV) method. This method involves selecting an optimal ε by minimizing a cost function that evaluates the errors of a series of partial RBF fits on the data points. The unknown error is estimated through a process where the data is divided into a *training* set, which includes all data points except one, and a *validation* set, which contains the single excluded point. The training data is used to construct the partial fit, while the validation point is used to compute the error. By repeating this process for each data point, a vector of error valuation is obtained. Then, the optimal value of ε is determined by using the cost function. For further details, see [46].

The LOOCV technique involve the evaluation, for each ε and for each $j \in \{1, \dots, n\}$, of the error

$$e_j(\varepsilon) = f(\boldsymbol{x}_j) - I_f^j(\boldsymbol{x}_j)$$

at the point $\boldsymbol{x}_j$ (validation) not used for the fitting of the partial RBF interpolant

$$I_f^j(\boldsymbol{x}) = \sum_{k=1,\ k\neq j}^{n} c_k \kappa_\varepsilon(\boldsymbol{x}, \boldsymbol{x}_k). \tag{3.1}$$

The interpolant I_f^j is fitted on $X_j = X \setminus \{x_j\}$ and $F_j = F \setminus \{f_j\}$ that are the training data and the data values respectively. Whilst, the coefficients c_k in (3.1) are determined

by interpolating only the set X_j, i.e.,

$$I_f^j(\boldsymbol{x}_k) = f(\boldsymbol{x}_k), \quad k = 1, \dots, j-1, j+1, \dots, n.$$

The optimal value of ε is found as

$$\varepsilon^* = \text{argmin}_\varepsilon ||\boldsymbol{e}(\varepsilon)||, \qquad \boldsymbol{e} = (e_1, \dots, e_n)^\intercal,$$

where $||\cdot||$ is any norm used in the minimization problem, for instance, the ∞-norm.

As can be seen from its formulation, the LOOCV is computationally expensive. To avoid this hitch a simplified error computation rule, proposed by Rippa in [98], can be used [98]

$$e_j(\varepsilon) = \frac{c_j}{(\mathsf{K}_\varepsilon^{-1})_{jj}}.$$

where c_j is the jth coefficient of the solution vector $\boldsymbol{c} = \mathsf{K}_\varepsilon^{-1}\boldsymbol{f}$ in (1.13), and $(\mathsf{K}_\varepsilon^{-1})_{jj}$ is the jth diagonal element of the inverse of the *full* RBF matrix K_ε. Notably, this approach requires solving only a single linear system for the entire data set X, thereby avoiding the need to solve n interpolation problems on $n-1$ points. This significantly reduces the computational cost from $O(n^4)$ to $O(n^3)$. However, it is important to mention that Rippa's formulation is not applicable when performing a least squares approximation, as the matrix K_ε becomes non-invertible in such cases. Thus, defining the the error function $Er(\varepsilon)$ as follows:

$$Er(\varepsilon) = \max_{j=1,\dots,n} \left| \frac{c_j}{(\mathsf{K}_\varepsilon^{-1})_{jj}} \right|. \tag{3.2}$$

The optimal value ε^* for the shape parameter is the one that minimizes the error function $Er(\varepsilon)$.

A good approximation of ε^* can be found evaluating the error function on a finite set of equally spaced values between 0 and ε_{max}, with ε_{max} large enough. The LOOCV technique involves evaluating the error function for each value in this discrete set, even if some values do not yield good results. Due to the discrete nature of the search, it is generally not possible to attain the global minimum. Additionally, the inversion of the interpolation matrix can introduce avoidable instabilities during computation.

A potential extension of LOOCV involves incorporating a univariate optimizer to guide the search for the optimal ε parameter. This approach could improve computational efficiency but would depend on the choice of the initial parameter value. Moreover, applying LOOCV with an optimizer to the approximation case would necessitate abandoning Rippa's formula, resulting in high computational costs that do not justify its use.

In the following section, we suggest a different technique that preserves computational expenses, circumvents poor choices of ε, and enables a continuous exploration of the parameter space. This method produces a finer approximation of ε^* and bypasses related possible failing. For a comprehensive discussion with examples of the challenges associated with searching for the shape parameter, see [43, Chapter 17].

3.2 Bayesian Optimization

When seeking a global maximizer for an unknown or complex to evaluate function g over a bounded domain X, *Bayesian optimization* [82] provides an elegant solution. Widespread

in machine learning, Bayesian optimization is an iterative method grounded in the principle of resource exploitation. This approach involves constructing a probabilistic model of g, known as the *surrogate model*, which aids in determining the sampling points within X through an acquisition function. As each iteration progresses, the model's distribution is updated and subsequently utilized in the following iteration. Although there is a computational cost associated with selecting the next evaluation point, this process is justified when evaluations of g are expensive. The goal is to achieve the maximum with minimal iterations, as seen in the optimization of error functions for costly machine learning training processes, such as multi-layer neural networks (refer to [3, 12]).

In this section, we briefly review the Bayesian optimization technique referring the reader to [17] for a more detailed description.

A *Gaussian Process* (GP) is a set of random variables where every finite subset has a multivariate Gaussian distribution. It is defined by a mean function $m : \mathcal{X} \to \mathbb{R}$ and a positive definite covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, with $\mathcal{X} \subseteq \mathbb{R}$ (see [96] for further details).

Gaussian processes are the preferred surrogate model for Bayesian optimization because they offer low evaluation costs and can incorporate prior beliefs about the objective function. When modeling the target function with a Gaussian process, denoted as $g(x) \sim \mathcal{GP}\big(m(x), k(x, x')\big)$, we impose the following conditions:

- $\mathbb{E}\big[g(x)\big] = m(x)$;
- $\mathbb{E}\big[\big(g(x) - m(x)\big)\big(g(x') - m(x')\big)\big] = k(x, x')$.

When making predictions based on some observations, the assumption of joint Gaussianity allows us to use the standard formula for the mean and variance of a conditional normal distribution. Suppose we have s observations $\boldsymbol{g} = (g(x_1), \ldots, g(x_s))^\intercal$ at points $\mathbf{x} = (x_1, \ldots, x_s)^\intercal$ and a new point $\bar{x}$ where we want to predict $\bar{g}$, the value of $g(\bar{x})$. The previous observations $\boldsymbol{g}$ and the predicted value $\bar{g}$ are jointly normally distributed:

$$Pr\left(\begin{bmatrix}\boldsymbol{g}\\ \bar{g}\end{bmatrix}\right) = \mathcal{N}\left[\begin{bmatrix}\mu(\mathbf{x})\\ \mu(\bar{x})\end{bmatrix}, \begin{bmatrix}K(X,X) & K(\mathbf{x},\bar{x})\\ K(\mathbf{x},\bar{x})^\intercal & k(\bar{x},\bar{x})\end{bmatrix}\right],$$

where X is the $s \times s$ matrix with (i,j)-element (x_i, x_j), $K(X,X)$ is the $s \times s$ matrix with (i,j)-element $k(x_i, x_j)$, and $K(\mathbf{x}, \bar{x})$ is a $s \times 1$ vector whose element i is given by $k(x_i, \bar{x})$. Since $Pr(\bar{g}|\boldsymbol{g})$ must also be normal, it follows that:

$$\begin{aligned} Pr(\bar{g}|\boldsymbol{g}) = \quad & \mathcal{N}[\mu(\bar{x})K(X,\bar{x})^\intercal K(X,X)^{-1}(\boldsymbol{g} - \mu(\mathbf{x})), k(\bar{x},\bar{x}) - \\ & -K(X,\bar{x})^\intercal K(X,X)^{-1}K(X,\bar{x})] \end{aligned}$$

This approach allows us to estimate the distribution, mean, and covariance at any point in the domain. When data locations and values obtained from evaluating the target function are provided to the model, they create a posterior distribution over functions. This posterior distribution is then used as the prior for the next iteration (see Figure 3.1). Specifically, when a function is modeled by a Gaussian Process (GP), observing a value at a particular point corresponds to observing the random variable associated with that point.

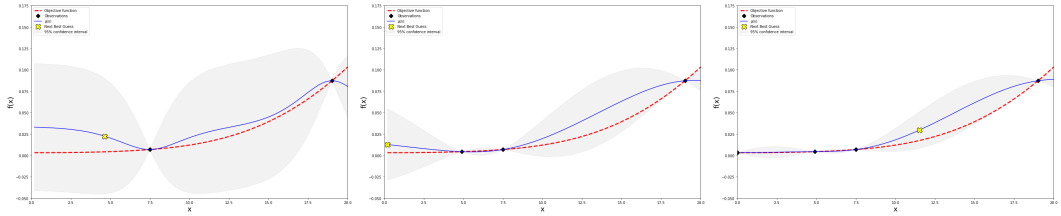

Figure 3.1: A demonstration of two steps of Bayesian optimization with two initial points. The figure on the left captures the state before the first Bayesian iteration, while the subsequent images illustrate the next two Bayesian steps. The cross highlighted in yellow marks the point chosen by the Expected Improvement acquisition function, which is then employed to refine the Gaussian model. Evidently, with each iteration, the process variance decreases, and the confidence interval tightens around the objective function.

3.3 Acquisition Functions

An acquisition function $a : \mathcal{X} \to \mathbb{R}$ serves to determine the next point for the objective function to evaluate. The selected point is the one that maximizes the acquisition function, and its evaluation by the objective function is then used to update the surrogate model (see Figure 3.1). An acquisition function is designed so that a high value indicates the potential for high values in the objective function. There is a trade-off between exploration and exploitation when choosing an acquisition function: exploration involves selecting points with high uncertainty, which are typically far from previously evaluated points, while exploitation involves selecting points near those already evaluated by the objective function. The most common acquisition functions include:

- **Probability of Improvement**: maximize the probability of improvement over the best current value;
- **Expected Improvement**: maximize the expected improvement over the current best;
- **GP Upper Confidence Bound**: minimize the cumulative regret[1] over the course of the optimization.

3.4 Expected Improvement

The acquisition function used in this work is the *Expected Improvement* (EI) function [63]. This function considers not only the probability that a candidate point will surpass the current maximum but also the magnitude of this improvement.

Assume that after several iterations, the current maximum of the objective function is $g(\hat{x})$. For a new point x, the Expected Improvement acquisition function calculates the

[1] Regret is a performance metric commonly used in Reinforcement Learning. In a maximization setting of a function g it represents the loss in rewards due to not knowing g's maximum points beforehand. If $x^* = \operatorname{argmax} g(x)$, the regret for a point x is $g(x^*) - g(x)$.

expected improvement $g(x) - g(\hat{x})$ over the portion of the normal distribution that exceeds the current maximum (see Figure 3.2):

$$EI(x) = \int_{g(\hat{x})}^{\infty} (g^*(x) - g(\hat{x})) \frac{1}{\sqrt{2\pi}\sigma(x)} e^{-\frac{1}{2}[(g^*(x)-\mu(x))/\sigma(x)]^2} dg^*(x), \tag{3.3}$$

where $g^*(x)$, $\mu(x)$ and $\sigma(x)$ represent the predicted value by the surrogate model, the expected value and the variance of x, respectively. Solving integral (3.3) it is possible to

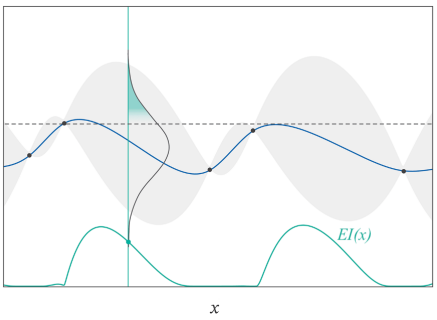

Figure 3.2: The black dots indicate where the objective function has been evaluated. The blue curve shows the mean $\mu(x)$ of the surrogate model, and the grey shaded region represents its 95% confidence interval. The teal curve depicts the Expected Improvement $EI(x)$.

retrieve the following closed form for the evaluation of the Expected Improvement:

$$EI(x) = \begin{cases} (\mu(x) - g(\hat{x}))\Phi(Z) + \sigma(x)\phi(Z), & \text{if } \sigma(x) > 0, \\ 0, & \text{if } \sigma(x) = 0, \end{cases} \tag{3.4}$$

where $Z = \frac{\mu(x)-g(\hat{x})}{\sigma(x)}$, while ϕ and Φ are the Probability Density Function and Cumulative Distribution Function of the standard normal distribution $\mathcal{N}(0,1)$. Moreover, the following extension of (3.4), proposed in [70], allows for a trade-off between exploration and exploitation through the use of a non-negative parameter ξ:

$$EI(x) = \begin{cases} (\mu(x) - g(\hat{x}) - \xi)\Phi(Z) + \sigma(x)\phi(Z), & \text{if } \sigma(x) > 0, \\ 0, & \text{if } \sigma(x) = 0, \end{cases} \tag{3.5}$$

where $Z = \frac{\mu(x)-g(\hat{x})-\xi}{\sigma(x)}$.
We refer the reader to chapter 4 for the pseudocode and further explanation of Bayesian optimization applied to the shape parameter search in RBF approximation.

Chapter 4

Bayesian Optimization for RBF Shape Parameter Tuning

Kernel-based methods are widely regarded as powerful tools for interpolating and approximating scattered data sets. Their popularity in approximation theory, particularly in meshfree approximation, stems from their ability to handle high-dimensional problems while maintaining a high level of accuracy in the resulting approximations. These techniques involve the use of a weighted sum of radial kernels or Radial Basis Functions (RBFs) (see Chapter 1), which are influenced by a parameter known as the *shape parameter*. This parameter plays a crucial role in determining the accuracy of the radial kernel method, making its selection critical. However, there is a well-known trade-off between accuracy and stability (see Section 1.5). This challenge has led many researchers to rely on trial-and-error methods, often resulting in empirical findings that can be both time-consuming and resource-intensive, or to adopt ad-hoc criteria [53, 60] that may not always yield optimal results. Other systematic methods for identifying the best value or an optimal range for the shape parameter have been suggested in [11, 114]. Although these approaches are generally applicable, they tend to be computationally demanding. The task of selecting an optimal shape parameter remains an active area of research in various fields of applied mathematics and scientific computing (see, for instance, [22, 32, 69]).

This chapter aims to introduce a versatile and effective method for approximation and interpolation that achieves accurate error estimates without requiring extensive trial-and-error processes. The core concept involves leveraging a well-established statistical method known as *Bayesian Optimization* (BO) [123] to identify the optimal shape parameter. BO, commonly employed in machine learning for optimizing complex or hard-to-evaluate functions, can be applied to hyper-parameter tuning to bypass the computation and assessment of parameters that are unlikely to be optimal. This approach significantly reduces computational time during the parameter search phase, enhancing efficiency.

To finalize this study, we will compare the results obtained using Bayesian Optimization with those achieved through Rippa's variant of the Leave One Out Cross Validation method (see Chapter 3). This method, which reduces the computational cost from n^4 to n^3 for n data points, offers a more efficient alternative to the standard approach originally introduced in [56]. Rippa's LOOCV is a widely recognized and well-established technique

for selecting an appropriate RBF shape parameter in data interpolation. It has since gained popularity in applied sciences, particularly for evaluating training performance [127, 128].

The rest of the chapter revisits topics covered in the published work [28], co-authored by the author of this thesis. Notably, section 4.1 explains in detail, with pseudocode, the Bayesian Optimization applied to the shape parameter search in RBF approximation, while, Sections 4.2 and 4.3 are devoted to numerical results using generated and real datasets respectively.

4.1 Algorithms

In this section, we will delve into the workings of Bayesian Optimization (BO) when applied to an approximation or interpolation process. Consider a set of points X_n with known associated data values F_n. We focus on a subset $\tilde{X}_s \subseteq X_n$ of the Radial Basis Function (RBF) centers. When $\tilde{X}_s$ is strictly a subset of X_n, we encounter an approximation problem in the least squares sense (see section 1.7). Conversely, if $\tilde{X}_s$ is equal to X_n, the scenario becomes one of interpolation. Bayesian Optimization helps in efficiently searching for the optimal shape parameter by constructing a probabilistic model of the objective function. This model guides the search process, prioritizing regions of the parameter space that are more likely to contain optimal values. By doing so, BO reduces the number of function evaluations needed, thus saving computational resources while still aiming for a high degree of accuracy in the approximation or interpolation.

Algorithm 1 is a pseudocode adaptation of the *BayesianOptimization library [88]. Specifically, it traces the optimisation process carried out by the* optimisation method of the *BayesianOptimisation class, which in turn uses the methods* fit and *predict of the function* GaussianProcessRegressor of *sklearn.gaussian_process package [91]. For the details about the implementation of* GaussianProcessRegressor we refer the reader to [96, Algorithm 2.1]. To perform the optimization, BO requires training and validation sets to evaluate the validation error while the optimization process in conducted. To fulfill this requirement we divide the sets $X_n, \tilde{X}_s$ and F_n into $X_{n_{train}}, X_{n_{val}}$, $\tilde{X}_{s_{train}}, \tilde{X}_{s_{val}}$ $F_{n_{train}}, F_{n_{val}}$ in such a way that

$$\begin{aligned} |X_{n_{train}}| &= \lfloor 0.8 \times |X| \rfloor, & |X_{n_{val}}| &= \lceil 0.2 \times |X| \rceil, \\ |\tilde{X}_{s_{train}}| &= \lfloor 0.8 \times |\tilde{X}| \rfloor, & |\tilde{X}_{s_{val}}| &= \lceil 0.2 \times |\tilde{X}| \rceil, \\ |F_{n_{train}}| &= \lfloor 0.8 \times |F| \rfloor, & |F_{n_{val}}| &= \lceil 0.2 \times |F| \rceil. \end{aligned}$$

During the optimization, the used measure for the accuracy of the approximant on the validation set is the Maximum Absolute Error (MAE) defined as follows:

$$\mathrm{MAE}_{X_{n_{val}}, F_{n_{val}}}(\mathbf{I_f}) = \max_{x_i \in X_{n_{val}}, f_i \in F_{n_{val}}} |I_f(x_i) - f_i|, \tag{4.1}$$

where $X_{n_{val}}$ is the validation set of data locations, $F_{n_{val}}$ contains the corresponding data values and $\mathbf{I_f} = (I_f(x_1), \ldots, I_f(x_{k_{val}}))$, with $k_{val} = |X_{n_{val}}|$. The optimization process is conducted on $g = -\mathrm{MAE}_{X_{n_{val}}, F_{n_{val}}}(\cdot)$ because the function is designed for maximization while we are searching for a minimum. In particular, given the points $X_{n_{val}}$ and a specific value of ε, it represents how close the approximation evaluated by **Algorithm** 2 are to the known values $F_{n_{val}}$.

Algorithm 1 BO$(X_{n_{train}}, \tilde{X}_{s_{train}}, F_{n_{train}}, X_{n_{val}}, g, \mathcal{X}, a, \xi, nstart, niter)$

Input:

$X_{n_{train}}$: data locations, $\tilde{X}_{s_{train}}$: RBF centers, $F_{n_{train}}$: data values, $X_{n_{val}}$: evaluation points, g: function to maximize, $\mathcal{X}$: parameter search interval, a: acquisition function, ξ: exploration-exploitation parameter, $nstart$: number of starting iterations, $niter$: number of Bayesian iterations.

$\boldsymbol{\varepsilon} \to$ sample of $nstart$ random parameter values in $\mathcal{X}$
$\mathbf{I}_{\mathbf{f}_{\varepsilon_j}} \to \mathbf{RBF}(X_{n_{train}}, \tilde{X}_{s_{train}}, F_{n_{train}}, X_{n_{val}}, \varepsilon_j),\ j = 1, \ldots, nstart$ (call to **Algorithm 2**)
$\boldsymbol{g} \to (g(\mathbf{I}_{\mathbf{f}_{\varepsilon_1}}), \ldots, g(\mathbf{I}_{\mathbf{f}_{\varepsilon_{nstart}}}))$
for $i = 1 : niter$ **do**
  Fit the Gaussian process on $(\boldsymbol{\varepsilon}, \boldsymbol{g})$
  Evaluate a on a set of random parameter values in $\mathcal{X}$
  Select the parameter $\hat{\varepsilon}$ that maximizes a
  $\mathbf{I}_{\mathbf{f}_{\hat{\varepsilon}}} \to \mathbf{RBF}(X_{n_{train}}, \tilde{X}_{s_{train}}, F_{n_{train}}, X_{n_{val}}, \hat{\varepsilon})$ (call to **Algorithm 2**)
  $\boldsymbol{\varepsilon} \to \boldsymbol{\varepsilon} \cup \hat{\varepsilon}$
  $\boldsymbol{g} \to \boldsymbol{g} \cup g(\mathbf{I}_{\mathbf{f}_{\hat{\varepsilon}}})$
end for
$\varepsilon^* \to \operatorname{argmax} \boldsymbol{g}$

Output:

ε^*: Best shape parameter.

Algorithm 2 RBF$(X, \tilde{X}, F, \bar{X}, \varepsilon)$

Input:

X: data locations, $\tilde{X}$: RBF centers, F: data values, $\bar{X}$: evaluation points, ε: shape parameter.

if $\tilde{X} = X$ **then**
  Solve the interpolation system (1.13)
else
  Solve the approximation system (1.15)
end if

Output:

$\mathbf{I}_{\mathbf{f}}$: Evaluation of the approximated or interpolated solution on $\bar{X}$.

The selection of the next value of ε to evaluate is determined by an acquisition function a and a surrogate model of the objective function g, which is generated by fitting a Gaussian process. In our case, the Expected Improvement (3.5) balances exploration and exploitation through the ξ parameter (refer to Subsection 3.2). It's important to note that the Gaussian process requires some initial iterations to be properly initialized.

Here, we begin with *niter* random values of ε within the search space $\mathcal{X} = (0, \varepsilon_{max}]$ along with their corresponding values of g. After this initialization, the optimization process, driven by the acquisition function, carries out *niter* optimization steps. During these steps, the Gaussian process is continuously updated and utilized in conjunction with the acquisition function. Ultimately, the value ε^* that maximizes the function g, which in this context minimizes the $\text{MAE}_{X_{n_{val}}, F_{n_{val}}}(\cdot)$, is identified as the optimal shape parameter.

Finally, with the optimal ε^* determined, an RBF fit is performed, leading to an approximate solution on the points of $\bar{X}$. For a comprehensive analysis of the complexity of Bayesian Optimization, we direct the reader to [103]. A summary of the complete RBF-BO algorithm is provided in **Algorithm** 3.

Algorithm 3 RBF-BO$(X_n, \tilde{X}_s, F_n, \bar{X}, \mathcal{X}, a, \xi, nstart, niter)$

Input:

X_n: data locations, $\tilde{X}_s$: RBF centers, F_n: data values, $\bar{X}$: evaluation points, $\mathcal{X}$: parameter search interval, a: acquisition function, ξ: exploration-exploitation parameter, *nstart*: number of starting iterations, *niter*: number of Bayesian iterations.

Split X_n, $\tilde{X}_s$ and F_n in $X_{n_{train}}, X_{n_{val}}$, $\tilde{X}_{s_{train}}, \tilde{X}_{s_{val}}$, $F_{n_{train}}, F_{n_{val}}$
Set $g \to -\text{MAE}_{X_{n_{val}}, F_{n_{val}}}(\cdot)$
$\varepsilon^* \to \mathbf{BO}(X_{n_{train}}, \tilde{X}_{s_{train}}, F_{n_{train}}, X_{n_{val}}, g, \mathcal{X}, a, \xi, nstart, niter)$ (call to **Algorithm** 1)
$\mathbf{I_f} \to \mathbf{RBF}(X_n, \tilde{X}_s, F_n, \bar{X}, \varepsilon^*)$ (call to **Algorithm** 2)

Output:

$\mathbf{I_f}$: Evaluation of the approximated or interpolated solution on $\bar{X}$.

4.2 Numerical Experiments

As described in section 3.2, the BO is presented as a maximization process and, since we are interested in the parameter ε who produces the minimum error, we consider as objective function the $-\text{MAE}_{X_{n_{val}}, F_{n_{val}}}(\cdot)$ both in interpolant (1.12) and approximant (1.14) framework.

The implementation of all the algorithms was developed in Python 3.9 utilizing the *BayesianOptimization* library [88] for the optimization tasks, in which the Matérn 5/2 is the default kernel used for the Gaussian process. The remainder of the section is organized in two subsection: the first presents numerical experiments related to RBF interpolation, while the second examines the algorithm's behaviour and performance during a gradual transition from approximation to interpolation. To evaluate the quality of the approximant, we consider a set of points with corresponding data values $(\bar{X}, \bar{F})$ that were not used in

fitting the approximant. We then define the MAE on $(\bar{X}, \bar{F})$ as follows:

$$\mathrm{MAE}_{\bar{X},\bar{F}}(\mathbf{I_f}) = \max_{x_i \in \bar{X}, f_i \in \bar{F}} |I_f(x_i) - f_i|, \tag{4.2}$$

where $\mathbf{I_f} = (I_f(x_1), \ldots, I_f(x_{\bar{k}}))$, with $\bar{k} = |\bar{X}|$.
All the tests reported in this section are performed using Python 3.9.12 on a 1.2 GHz Quad-Core Intel Core i7 processor, 16 GB 3733 MHz LPDDR4X RAM MacBook Air (2020).

4.2.1 Interpolation Results

To make a comparison, we use LOOCV to assess both computation time and error. The two methods, LOOCV and BO, differ significantly in their approaches: LOOCV calculates the error based on a single point, whereas BO evaluates the error across a set of points. In our experiments, we use two distinct data sets: (X_n, F_n) for fitting the approximant and $(\bar{X}, \bar{F})$ for evaluating the approximation error, with the latter remaining constant throughout the experiments. The use of the set (X_n, F_n) varies depending on the method employed for selecting the shape parameterε. For LOOCV, we rely on Rippa's rule to compute the LOOCV errors, fitting the interpolant on the entire set X, without dividing it into training and validation subsets. Instead, when using BO, we further divide (X_n, F_n) by selecting 80% of the points for the training set, while the remaining 20% constitutes the validation set, where the error is calculated during optimization. This results in the subsets $X_{n_{train}}, X_{n_{val}}$, $F_{n_{train}}, F_{n_{val}}$, as previously described in Section 4.1. After determining the optimal shape parameter for both methods, we fit a radial kernel interpolant for each on (X_n, F_n) and evaluate the error on $(\bar{X}, \bar{F})$. This approach gives LOOCV an advantage since the same set of data points and values is used both for parameter determination and for fitting the final model. For LOOCV, the errors were computed across 500 equally spaced values of ε within the search space $\mathcal{X} = (0, \varepsilon_{max}] = (0, 20]$. For BO, we performed 5 random initial iterations followed by 25 optimization steps. Additionally, the exploration-exploitation trade-off was managed using the ξ parameter (see Section 3.2), with three values considered: $\xi = 0.1$ (favoring exploration), $\xi = 0.01$, and $\xi = 0.001$ (favoring exploitation). Furthermore, to have an additional comparison, we consider a variant of the LOOCV called LOOCV*, which incorporates a univariate optimizer. Instead of evaluating all 500 ε values, LOOCV* employs the *minimize* function from the *scipy.optimize* library [126] to guide the parameter search. The *minimize* function was applied in its default form, starting with a value of 10, the median of the parameter search space $\mathcal{X}$.

The experiments are carried out using as data location random and Halton points in the domain $\Omega = [0, 1]^2$ each with 3 different sizes n. We use 3 different RBFs and 2 test functions listed below.
RBF:

$$\varphi(\varepsilon r) = \begin{cases} \exp(-\varepsilon^2 r^2), & \text{Gaussian } C^\infty \quad \text{(GA)} \\ \exp(-\varepsilon r)(\varepsilon r + 1), & \text{Matérn } C^2 \quad \text{(M2)} \\ \max\left(1 - \varepsilon r, 0\right)^4 (4\varepsilon r + 1), & \text{Wendland } C^2 \quad \text{(W2)} \end{cases}$$

Test functions [24, 97]:

$$f_1(\boldsymbol{x}) = 0.75\exp\left[-\frac{(9x_1-2)^2}{4} - \frac{(9x_2-2)^2}{4}\right] + 0.75\exp\left[-\frac{(9x_1-2)^2}{49} - \frac{9x_2+1}{10}\right] +$$
$$+0.5\exp\left[-\frac{(9x_1-7)^2}{4} - \frac{(9x_2-3)^2}{4}\right] - 0.2\exp\left[-(9x_1-4)^2 - (9x_2-7)^2\right],$$
$$f_2(\boldsymbol{x}) = \frac{\sqrt{64-81((x_1-0.5)^2+(x_2-0.5)^2)}}{9} - 0.5.$$

We highlight that the kernels GA and M2 are globally supported and the kernel W2 is compactly supported. The data in Tables 4.1–4.6 generally indicate that the errors from LOOCV and BO are quite similar, with the primary distinction being computational time. BO tends to be faster, especially for larger values of n, where the cost of inverting the interpolation matrix becomes significant. In instances where BO does not match the precision of LOOCV, this is often due to the lower regularity of the function being approximated, which can be mitigated by increasing the number of Bayesian iterations. Additionally, a parameter value of $\xi = 0.01$ is effective in most scenarios. For $n = 1000$ and $n = 500$, the results from LOOCV* are comparable to those from BO. Although LOOCV* sometimes requires less time, BO generally achieves better accuracy, possibly because LOOCV* can get trapped in local minima. For $n = 250$, LOOCV* performs better than BO, which is limited by a fixed number of iterations during space exploration. The LOOCV* method, which uses a stop tolerance, can reduce computational costs. Overall, while both methods are comparable, BO is preferable for problems with a large number of data points due to its faster convergence and independence from initial values.

			Matérn kernel			Gaussian kernel		
n	method	ξ	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	LOOCV		6.87e+01	6.76e-04	1.520000	7.12e+01	8.53e-05	7.040000
	LOOCV*		1.97e+00	3.23e-03	9.990318	5.75e+00	4.47e-04	10.000001
	BO	0.1	5.32e+00	6.65e-04	1.147200	4.98e+00	3.40e-04	6.159917
		0.01	6.18e+00	6.64e-04	1.129313	4.41e+00	1.66e-03	7.042178
		0.001	4.33e+00	6.63e-04	0.788350	4.33e+00	1.45e-04	6.465295
500	LOOCV		1.04e+01	2.81e-03	1.680000	1.10e+01	7.40e-04	5.920000
	LOOCV*		6.08e-01	2.81e-03	1.705232	4.01e+00	1.57e-04	6.346624
	BO	0.1	1.90e+00	2.73e-02	0.001233	2.36e+00	1.14e-02	5.656844
		0.01	1.72e+00	2.83e-03	1.453144	2.45e+00	1.99e-01	5.663782
		0.001	1.98e+00	2.60e-02	0.001000	3.30e+00	1.10e-03	5.906976
250	LOOCV		2.23e+00	4.50e-03	0.840000	2.11e+00	4.74e-02	5.520000
	LOOCV*		7.60e-01	4.51e-03	0.818112	5.33e-01	3.72e-02	6.293530
	BO	0.1	1.52e+00	4.35e-03	1.147200	1.80e+00	5.00e-02	5.475028
		0.01	1.48e+00	4.20e-03	1.528790	1.86e+00	3.51e-02	6.069408
		0.001	1.54e+00	4.19e-03	1.570866	2.55e+00	4.44e-02	6.630227

Table 4.1: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using M2 and GA kernels for the interpolation of f_1 on various sets of random points by LOOCV, LOOCV* and BO.

			Matérn kernel			Gaussian kernel		
n	method	ξ	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	LOOCV		6.09e+01	7.69e-04	1.240000	6.60e+01	1.29e-05	6.440000
	LOOCV*		3.87e+00	4.05e-04	1.251510	6.11e+00	3.72e-04	10.000009
	BO	0.1	4.72e+00	7.70e-04	1.147200	4.42e+00	2.91e-05	6.153352
		0.01	6.06e+00	7.68e-04	1.604766	4.31e+00	2.93e-05	6.847236
		0.001	6.20e+00	7.68e-04	1.328962	4.29e+00	9.71e-06	7.491428
500	LOOCV		7.59e+00	3.33e-03	4.640000	7.96e+00	1.08e-04	6.080000
	LOOCV*		5.13e-01	2.63e-03	1.495576	1.81e+00	1.56e-03	6.782924
	BO	0.1	1.54e+00	2.32e-03	1.147200	2.81e+00	1.06e-04	6.064566
		0.01	1.58e+00	2.32e-03	1.147200	2.55e+00	1.61e-04	6.116520
		0.001	1.61e+00	3.32e-03	0.001000	2.19e+00	1.20e-04	6.075517
250	LOOCV		1.24e+00	6.52e-03	2.280000	1.21e+00	5.82e-03	5.800000
	LOOCV*		3.82e-01	9.87e-03	3.365488	5.26e-01	2.58e-02	4.997309
	BO	0.1	1.49e+00	7.61e-03	3.639104	1.63e+00	6.31e-03	5.957693
		0.01	1.49e+00	7.61e-03	3.639104	2.12e+00	5.93e-03	5.844204
		0.001	1.39e+00	8.06e-03	3.981307	2.59e+00	5.90e-03	5.834172

Table 4.2: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using M2 and GA kernels for the interpolation of f_1 on various sets of Halton points by LOOCV, LOOCV* and BO.

			Matérn kernel			Gaussian kernel		
n	method	ξ	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	LOOCV		7.05e+01	8.60e-04	0.040000	7.90e+01	2.66e-05	6.000000
	LOOCV*		3.73e+00	1.18e-03	0.668190	7.16e+00	2.91e-04	10.000001
	BO	0.1	3.67e+00	1.41e-03	1.147200	4.28e+00	1.50e-05	4.349624
		0.01	3.69e+00	1.59e-03	1.507068	4.62e+00	7.28e-04	3.398101
		0.001	3.70e+00	1.08e-03	0.474564	5.20e+00	2.07e-05	4.782848
500	LOOCV		1.11e+01	2.24e-03	0.040000	1.10e+01	4.14e-05	3.840000
	LOOCV*		8.44e-01	2.75e-03	0.421860	1.96e+00	1.15e-03	7.824069
	BO	0.1	1.78e+00	8.53e-03	0.001233	1.77e+00	7.23e-06	3.121217
		0.01	1.77e+00	8.53e-03	0.001233	2.13e+00	7.23e-06	3.121217
		0.001	1.65e+00	2.71e-03	0.393487	2.66e+00	7.23e-06	3.121217
250	LOOCV		2.09e+00	1.20e-02	0.040000	2.09e+00	2.17e-04	2.760000
	LOOCV*		2.48e-01	1.27e-02	0.143446	4.90e-01	7.55e-04	1.136128
	BO	0.1	1.32e+00	3.07e-02	7.174972	2.53e+00	3.81e-04	1.867692
		0.01	1.40e+00	3.07e-02	7.174972	1.77e+00	9.57e-05	2.728965
		0.001	1.44e+00	1.13e-02	0.001000	2.04e+00	4.43e-04	2.979949

Table 4.3: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using M2 and GA kernels for the interpolation of f_2 on various sets of random points by LOOCV, LOOCV* and BO.

4.2.2 Approximation Results

In this subsection, we present experiments on approximation by varying the percentage of data locations in X_n used as approximation centers. When 100% of the nodes are used as centers, we essentially perform interpolation, yielding results similar to those in Subsection 4.2.1, though not identical due to the stochastic nature of the experiments. Generally, opting for approximation over interpolation results in a loss of accuracy but offers time

			Matérn kernel			Gaussian kernel		
n	method	ξ	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	LOOCV		5.23e+01	2.61e-04	0.040000	5.07e+01	1.44e-06	4.320000
	LOOCV*		3.84e+00	3.35e-04	0.257555	6.11e+00	1.21e-04	9.999999
	BO	0.1	3.10e+00	4.48e-04	1.147200	3.19e+00	1.79e-06	3.121217
		0.01	3.30e+00	3.94e-04	0.838285	3.68e+00	1.91e-06	3.402787
		0.001	3.37e+00	4.11e-04	0.934542	3.87e+00	1.79e-06	3.121217
500	LOOCV		8.49e+00	7.10e-04	0.040000	7.46e+00	2.10e-05	4.840000
	LOOCV*		6.22e-02	7.01e-03	9.999965	1.17e+00	3.06e-04	5.917107
	BO	0.1	1.51e+00	1.12e-03	1.147200	1.69e+00	7.76e-06	2.199642
		0.01	1.51e+00	6.61e-04	0.001233	2.54e+00	1.70e-05	3.378652
		0.001	1.52e+00	1.22e-03	0.001000	2.66e+00	1.43e-05	3.377515
250	LOOCV		1.23e+00	3.73e-03	0.040000	1.20e+00	4.09e-05	2.240000
	LOOCV*		4.21e-01	2.43e-03	0.020730	5.96e-01	6.39e-04	4.251622
	BO	0.1	1.37e+00	3.64e-03	0.001233	2.74e+00	1.70e-03	1.414414
		0.01	1.39e+00	3.64e-03	0.001233	2.32e+00	3.03e-05	2.057707
		0.001	1.34e+00	8.87e-03	3.121217	1.67e+00	4.20e-05	2.820584

Table 4.4: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using M2 and GA kernels for the interpolation of f_2 on various sets of Halton points by LOOCV, LOOCV* and BO.

			f_1			f_2		
n	method	ξ	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	LOOCV		9.37e+01	6.77e-04	0.280000	7.77e+01	9.13e-04	0.040000
	LOOCV*		1.33e+01	6.71e-04	0.240126	8.86e+00	8.95e-04	0.030410
	BO	0.1	4.86e+00	7.37e-04	0.001000	5.35e+00	8.34e-04	0.001000
		0.01	5.87e+00	6.85e-04	0.001000	4.97e+00	8.30e-04	0.001000
		0.001	6.07e+00	6.85e-04	0.001000	6.59e+00	8.30e-04	0.001000
500	LOOCV		1.25e+01	2.95e-03	0.280000	1.24e+01	2.38e-03	0.040000
	LOOCV*		1.59e+00	2.95e-03	0.280290	1.66e+00	5.08e-03	0.818422
	BO	0.1	1.86e+00	3.44e-03	0.001000	1.82e+00	2.18e-03	0.001000
		0.01	2.09e+00	3.49e-03	0.001000	1.87e+00	2.19e-03	0.001000
		0.001	2.49e+00	3.49e-03	0.001000	2.33e+00	2.21e-03	0.001096
250	LOOCV		2.82e+00	4.69e-03	0.320000	2.53e+00	1.27e-02	0.040000
	LOOCV*		1.63e+00	4.68e-03	0.287048	3.15e-01	1.27e-02	0.037173
	BO	0.1	1.56e+00	4.82e-03	0.440938	1.54e+00	2.34e-02	1.249925
		0.01	1.56e+00	4.68e-03	0.280038	1.41e+00	2.34e-02	1.237074
		0.001	2.41e+00	4.68e-03	0.270527	1.61e+00	2.33e-02	1.262232

Table 4.5: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using W2 kernel for the interpolation of f_1 and f_2 on various sets of random points by LOOCV, LOOCV* and BO.

savings. Tables 4.7–4.9 illustrate three scenarios:

1. **Small Training Set ($n = 250$ points)**: Precision is low across all cases, and there is minimal time saving with approximation. Therefore, interpolation is recommended to maintain accuracy.

2. **Medium Training Set ($n = 500$ points)**: In most cases, using 80% of the points as centers for approximation saves computational time compared to interpolation,

			f_1			f_2		
n	method	ξ	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	LOOCV		5.86e+01	7.91e-04	0.240000	5.96e+01	2.76e-04	0.040000
	LOOCV*		6.14e+00	4.58e-04	0.179607	1.84e+01	3.21e-04	0.046904
	BO	0.1	3.59e+00	8.25e-04	0.001000	3.25e+00	2.56e-04	0.001233
		0.01	3.61e+00	7.91e-04	0.243379	3.29e+00	2.57e-04	0.001000
		0.001	4.12e+00	7.93e-04	0.277008	4.20e+00	2.55e-04	0.001096
500	LOOCV		8.77e+00	3.81e-03	1.000000	8.88e+00	7.48e-04	0.040000
	LOOCV*		2.07e+00	3.09e-03	0.263759	2.29e+00	1.69e-03	0.019589
	BO	0.1	1.65e+00	2.53e-03	0.001000	1.69e+00	6.96e-04	0.001000
		0.01	1.84e+00	2.53e-03	0.001792	1.71e+00	6.96e-04	0.001000
		0.001	2.18e+00	2.53e-03	0.001096	2.13e+00	6.97e-04	0.001096
250	LOOCV		1.58e+00	6.67e-03	0.360000	1.53e+00	3.94e-03	0.040000
	LOOCV*		2.46e-01	1.18e-02	1.064206	9.70e-01	2.46e-03	0.011255
	BO	0.1	1.41e+00	7.18e-03	0.568376	1.47e+00	3.65e-03	0.001000
		0.01	1.99e+00	6.81e-03	0.449130	1.44e+00	3.65e-03	0.001000
		0.001	1.94e+00	6.70e-03	0.379231	1.84e+00	3.65e-03	0.001000

Table 4.6: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using W2 kernel for the interpolation of f_1 and f_2 on various sets of Halton points by LOOCV, LOOCV* and BO.

making it the preferred choice.

3. **Large Training Set ($n = 1000$ points)**: Significant time savings are observed when using 80% of the points as centers for approximation, without compromising precision. This strongly supports choosing approximation.

In summary, for larger datasets, the computational time savings become more substantial, offering a reasonable trade-off between computational expense and accuracy.

4.3 Application to Real Data

To evaluate the algorithm's performance in a real-world scenario, we utilize the *volcano* dataset from the statistical software package R [95], which includes 5307 elevation measurements from Maunga Whau (Mt. Eden) in Auckland, NZ, sampled on a $10m \times 10m$ grid. To create a comparable example to those in the previous section, we select a random subset of 1500 points, divided into a training set (X_n, F_n) of 1000 points and a test set $(\bar{X}, \bar{F})$ of 500 points (see Figure 4.1). As described in Section 4.1, (X_n, F_n) is further split into $X_{n_{train}}, X_{n_{val}}, F_{n_{train}}, F_{n_{val}}$ when using BO.

For this experiment, we focus on the interpolation case by setting $\tilde{X} = X$ and its divisions accordingly. We use a fixed parameter $\xi = 0.01$ and the M2 and W2 kernels. The results are presented in Table 4.10, and the resulting interpolant surfaces are shown in Figure 4.2. We restrict our analysis to the M2 and W2 kernels, excluding the GA kernel due to the low regularity of the real measurement data, which is a critical requirement for the efficient use of the GA kernel. Additionally, to ensure comparability, we compute both the $\text{MAE}_{\bar{X},\bar{F}}(\cdot)$ and its relative version, $\text{RMAE}_{\bar{X},\bar{F}}(\cdot)$, by dividing the $\text{MAE}_{\bar{X},\bar{F}}(\cdot)$ by the measurement with the highest absolute value.

		Random			Halton		
n	centers(%)	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	20	3.79e+00	2.96e-03	4.861528	2.98e+00	1.88e-03	5.147405
	40	5.59e+00	1.50e-03	5.285210	2.90e+00	1.05e-04	5.594780
	60	5.46e+00	2.33e-04	5.744282	3.92e+00	2.78e-05	6.348615
	80	5.85e+00	3.90e-05	6.319368	5.45e+00	2.43e-05	5.698942
	100	7.61e+00	7.87e-05	6.319614	6.35e+00	9.87e-06	6.309349
500	20	1.64e+00	1.98e-02	4.471739	1.57e+00	1.67e-02	5.361774
	40	2.48e+00	6.83e-03	5.289607	1.83e+00	1.68e-03	5.171920
	60	2.82e+00	2.89e-03	5.869333	2.33e+00	5.12e-04	5.336932
	80	2.60e+00	5.61e-03	5.043093	2.07e+00	6.12e-04	5.312729
	100	4.33e+00	2.34e-03	6.297596	2.20e+00	5.31e-04	5.459864
250	20	1.35e+00	7.73e-02	3.121217	1.43e+00	5.63e-02	3.772420
	40	1.38e+00	2.08e-02	4.491812	1.32e+00	1.87e-01	1.694190
	60	1.38e+00	2.70e-02	5.093333	1.28e+00	1.28e-02	4.491812
	80	1.55e+00	6.55e-02	4.548724	2.23e+00	1.04e-02	5.183731
	100	1.45e+00	1.34e-02	5.582551	1.36e+00	6.86e-03	5.143759

Table 4.7: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using GA kernel for the approximation of f_1 on various sets of random and Halton points as the center percentage varies by applying LOOCV and BO.

		Random			Halton		
n	centers(%)	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	20	3.79e+00	2.96e-03	4.861528	2.98e+00	1.88e-03	5.147405
	40	3.21e+00	2.01e-03	1.498876	2.95e+00	2.75e-03	2.164716
	60	3.60e+00	1.74e-03	1.312339	3.79e+00	1.51e-03	2.244381
	80	5.22e+00	1.02e-03	1.332212	5.78e+00	9.02e-04	1.676676
	100	6.41e+00	1.03e-03	1.332212	7.83e+00	8.61e-04	1.695884
500	20	3.49e+00	3.29e-02	7.491428	2.61e+00	1.64e-02	2.245995
	40	2.35e+00	1.16e-02	1.251539	2.96e+00	4.82e-03	2.087647
	60	3.77e+00	1.08e-02	2.086028	4.79e+00	5.07e-03	1.086503
	80	2.43e+00	4.36e-03	1.293058	4.43e+00	2.88e-03	2.647202
	100	2.85e+00	4.19e-03	1.398220	5.37e+00	2.51e-03	2.183219
250	20	1.34e+00	9.87e-02	0.488346	1.45e+00	4.99e-02	5.536034
	40	1.52e+00	6.54e-02	6.125847	1.26e+00	1.77e-02	2.069197
	60	3.18e+00	2.39e-02	2.085691	1.32e+00	9.46e-03	2.143143
	80	3.77e+00	1.66e-02	0.898543	2.51e+00	4.68e-03	1.521938
	100	2.86e+00	1.34e-02	4.262523	2.41e+00	4.44e-03	1.556272

Table 4.8: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using M2 kernel for the approximation of f_1 on various sets of random and Halton points as the center percentage varies by applying LOOCV and BO.

		Random			Halton		
n	centers(%)	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	ε^*
1000	20	3.89e+00	8.69e-03	0.293414	1.80e+00	1.30e-02	0.525864
	40	3.84e+00	2.16e-03	0.279970	3.35e+00	6.61e-03	0.382713
	60	5.80e+00	6.83e-03	0.286352	3.89e+00	3.89e-03	0.294302
	80	7.76e+00	6.66e-03	0.266835	5.52e+00	8.00e-04	0.320220
	100	8.81e+00	1.02e-03	0.310954	7.05e+00	8.59e-04	0.328610
500	20	2.23e+00	2.49e-02	0.127089	2.18e+00	3.22e-02	0.344410
	40	2.59e+00	8.41e-03	0.199470	2.14e+00	5.58e-03	0.337803
	60	3.40e+00	8.91e-03	0.323238	2.15e+00	4.47e-03	0.299074
	80	2.64e+00	6.81e-03	1.135019	2.42e+00	2.68e-03	0.261940
	100	2.98e+00	6.74e-03	1.120099	2.10e+00	2.60e-03	0.250589
250	20	1.26e+00	7.70e-02	1.065228	1.38e+00	7.97e-02	1.714649
	40	1.51e+00	4.42e-02	0.311686	1.54e+00	3.12e-02	0.368420
	60	1.53e+00	2.38e-02	0.736349	1.41e+00	1.67e-02	0.401290
	80	1.61e+00	1.76e-02	0.425162	1.50e+00	7.12e-03	0.311549
	100	1.74e+00	1.60e-02	0.423409	1.72e+00	4.63e-03	0.281553

Table 4.9: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$ and shape parameter using W2 kernel for the approximation of f_1 on various sets of random and Halton points as the center percentage varies by applying LOOCV and BO.

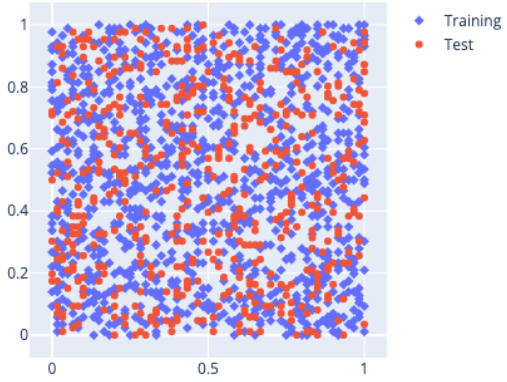

Figure 4.1: In red the training data and in blue the test data of the *volcano* dataset.

4.4 Discussion on results

When comparing parameter searches for interpolation using LOOCV and BO, the error values are generally of similar magnitude. However, the key difference lies in the computation time, with BO typically being an order of magnitude faster. For approximation, there is a balance between computational cost and precision. Specifically, using a set of centers equal to 80% of the nodes usually reduces computation time without significantly affecting precision. Further clarification is needed when LOOCV yields a better error value. This

kernel	method	time (s)	$\text{MAE}_{\bar{X},\bar{F}}$	$\text{RMAE}_{\bar{X},\bar{F}}$	ε^*
M2	LOOCV	9.66e+01	3.894942	0.020392	8.560000
	BO	6.38e+00	3.892905	0.020382	13.01788
W2	LOOCV	1.04e+02	3.886987	0.020351	1.600000
	BO	8.24e+00	3.860059	0.020210	2.291239

Table 4.10: Computational time, $\text{MAE}_{\bar{X},\bar{F}}$, $\text{RMAE}_{\bar{X},\bar{F}}$ and shape parameter using M2 and W2 kernels by LOOCV and BO.

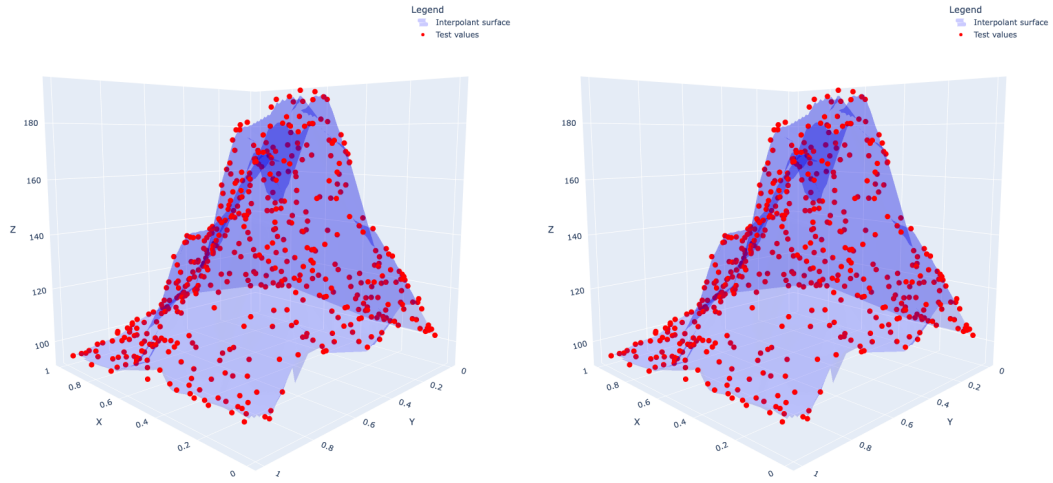

Figure 4.2: Interpolant surfaces and test values obtained using the M2 kernel (left) and the W2 kernel (right).

issue often arises from the limited number of BO iterations, which can cause the search to get stuck in local minima. Increasing the number of iterations and using an exploration-focused parameter ξ can address this problem. The performance of BO also depends on the choice of the covariance function and the prior distribution. In the implementation referenced in [88], the default covariance function is the Matérn 5/2 kernel [43], which assumes a relatively smooth function. Exploring different parameters for the covariance function or using alternative kernels could further enhance performance.
An immediate extension of the work presented in this chapter, and the focus of the next, is the application of BO with the Partition of Unity scheme. This approach aims to simultaneously determine the shape parameters of the Radial Basis Functions and the radius of the PU subdomains by optimising them in a 2-dimensional space, achieving optimal accuracy and saving computational time compared to individual optimisations.

Chapter 5

Bayesian Optimization for simultaneous shape parameter and radius search in RBF-PU Method

Radial basis function (RBF) interpolation and approximation have gained significant popularity in recent decades as a versatile tool for solving advanced problems in fields such as finance and engineering [10, 30, 43, 51, 109]. This increasing adoption is largely due to the method's simplicity in high-dimensional spaces, its strong convergence properties, and its flexibility in adapting to different geometric configurations [133]. However, as RBF methods are applied to increasingly complex, high-dimensional datasets, the computational cost rises, particularly due to the need to invert the collocation matrix in global interpolation problems. Additionally, high-dimensional interpolation matrices can lead to unstable and ill-conditioned computations (see Section 1.5 for more details). To address these challenges, a precise and efficient solution involves decomposing the global domain into overlapping subdomains. Each subdomain focuses on a smaller subproblem, and the results are combined using a partition of unity method with radial basis function interpolation (RBF-PUM). The core idea is to aggregate local approximations, each computed by solving small RBF interpolation problems within a subdomain, into a global partition of unity solution (See Chapter 3 for a detailed explanation of this method and its error analysis in the native space of the underlying RBFs). This localization of the approximation process, achieved through domain decomposition into smaller subproblems, has become widely applied across various fields of computational mathematics and scientific computing [5, 22, 27, 29]. In this chapter, we propose an RBF-PUM method that is formulated as a weighted sum of local RBF interpolants, with each interpolant depending on a shape parameter ε and a subdomain radius δ. Both of these parameters vary across subdomains. To optimize ε and δ simultaneously for each subdomain, we perform a multivariate search using Bayesian Optimization (BO) (see Chapter 3 for further details on BO).

Initially developed for machine learning to optimize complex and computationally expensive functions, BO is particularly suited for hyperparameter tuning tasks, such as the (ε, δ) search in RBF-PUM interpolation. It reduces computational costs by avoiding the evaluation of error functions for parameter sets that are unlikely to be optimal. We present and analyze the algorithms involved in this optimization process, demonstrating their effectiveness in terms of both time efficiency and accuracy. Moreover, we validate our approach by testing it on real-world benchmark datasets, such as the Tonga Trench and Franke's glacier. The remainder of the chapter revisits work published in [31], co-authored by the author of this thesis. Section 5.1 presents the algorithms, including pseudocode, with detailed discussions. Section 5.2 is dedicated to analyzing the complexity of the algorithms introduced in Section 5.1, while Sections 5.3 and 5.4 focus on numerical experiments involving test functions and real-world datasets, respectively.

5.1 Algorithms

This section is devoted to the description of the algorithms for RBF-PUM interpolation combined with Bayesian optimization. Let us suppose to have an interpolation problem like Problem 1.1.1, i.e. a set of points X_n, a set of values F_n and a set of evaluation points $\bar{X} = \{\bar{\boldsymbol{x}}_i, i = 1, \dots, \bar{n}\}$ on which we are interested in determining an approximate solution. The (ε, δ)-optimization and interpolation with partial results combination is performed by Algorithm 5 by invoking subroutines. Notably, the parameter optimization is achieved through Bayesian optimization by calling the Algorithm 8 while, the partition of unity for the evaluation of the approximant by calling the Algorithm 10. Particularly, after the determination of the optimal shape parameter-radii couple (ε, δ) for each subdomain by means of the Bayesian optimization the Algorithm 5 builds the approximations for the points in $\bar{X}$. To begin, we first determine the number of points in X, denoted as n, and the dimension of the space, denoted as m. A suitable number of subdomains is given by $\lfloor \frac{n}{2^m} \rfloor$ (see [43]). These values are then used to calculate the number of centers for the partition of unity, which are constructed as an equally spaced grid in the domain Ω. We focus on the specific case where $\Omega = [0,1]^m$, as any domain can be transformed into this one via a linear transformation. The distance tree for X_n is then evaluated using the *KDTree* function from the *scipy.spatial* package, and its method is employed to perform the search over the points. Subsequently, in each subdomain, the radius that guarantees the minimum density is determined through Algorithm 7. This value, $\boldsymbol{\delta}_{start}$, will later be applied during Bayesian optimization (Algorithm 8) to refine both the shape parameter and the radius within each subdomain. Finally, the RBF-PUM approximant is trained using the identified parameters (Algorithm 9), and the approximated function values are returned for the set of points $\bar{X}$.

The core of the process is handled by Algorithm 8, which is an adaptation of the Python library *BayesianOptimization* [88]. Specifically, the optimization process provided by the *optimisation* method of the *BayesianOptimisation* class is traced. This method sequentially utilizes the *fit* and *predict* methods from the *GaussianProcessRegressor* function within the *sklearn.gaussian_process* package [91]. Further implementation details for *GaussianProcessRegressor* can be found in [96, Algorithm 2.1].

To evaluate the quality of the approximant, the Maximum Absolute Error (MAE), the

Algorithm 5 BO-PUM$(X_n, F_n, \bar{X}, T, a, \xi, start, niter, min_{pts}, \tau, w)$

Input:

X_n: data points, F_n: data values, $\bar{X}$: evaluation points, I: ε search interval, a: acquisition function, ξ: exploration-exploitation parameter, $nstart$: number of starting points, $niter$: number of Bayesian iterations, min_{pts}: minimum number of points in a subdomain, τ: tolerance of the error during the parameters search, w: weight function.

$n \to$ number of points in X_n
$m \to$ space dimension of X_n
$d \to \lfloor \frac{n}{2^m} \rfloor$
$centers \to$ grid of d points in $\Omega = [0,1]^m$
$\boldsymbol{\varepsilon} \to$ 0-vector of length d
$\boldsymbol{\delta} \to$ 0-vector of length d
$DT_{X_n} \to$ distance tree of X_n
$\boldsymbol{\delta}_{start} \to$ **FIND-MIN-RADIUS**$(X_n, centers, d, m, min_{pts}, DT_{X_n})$
for $i = 1 : |centers|$ **do**
  $J \to [\boldsymbol{\delta}_{start}[i], 2\boldsymbol{\delta}_{start}[i]]$
  $[\varepsilon, \delta] \to$ **BO**$(X_n, F_n, I, J, a, \xi, nstart, niter, DT_{X_n}, centers[i], \tau)$
  $\boldsymbol{\varepsilon}[i] \to \varepsilon$
  $\boldsymbol{\sigma}[i] \to \sigma$
end for
$\mathbf{I_f} \to$ **PUM**$(X_n, F_n, \bar{X}, centers, DT_{X_n}, w, \boldsymbol{\varepsilon}, \boldsymbol{\delta})$

Output:

$\mathbf{I_f}$: evaluation of the interpolated solution on $\bar{X}$

Algorithm 7 FIND-MIN-RADIUS$(X_n, centers, d, m, min_{pts}, DT_{X_n})$

Input:

X_n: data points, $centers$: PU centers, d: number of centers, m: space dimension, min_{pts}: minimum number of points in a subdomain, DT_{X_n}: distance tree of X_n.

$\boldsymbol{\delta}_{start} \to \frac{\sqrt{m}}{2d^{\frac{1}{m}}} \times$ 1-vector of length $|centers|$
for $i = 1 : d$ **do**
  $X_{n_{sub}} \to$ retrieve the subset of X_n within distance $\boldsymbol{\delta}_{start}[i]$ from $centers[i]$ using DT_{X_n}
  while $|X_{n_{sub}}| < min_{pts}$ **do**
    $\boldsymbol{\delta}_{start}[i] \to \boldsymbol{\delta}_{start}[i] + \frac{1}{8}\frac{\sqrt{m}}{2d}$
    $X_{n_{sub}} \to$ retrieve the subset of X_n within distance $\boldsymbol{\delta}_{start}[i]$ from $centers[i]$ using DT_{X_n}
  end while
end for

Output:

$\boldsymbol{\delta}_{start}$: vector of subdomain radii that ensure the minimum densities.

Relative Maximum Absolute Error (RMAE), and the Relative Root Mean Squared Error (RRMSE) are introduced, defined as follows:

$$\mathrm{MAE}(X, F, \mathbf{I_f}) = \mathrm{MAE}_{X,F}(\mathbf{I_f}) = \max_{\boldsymbol{x}_i \in X, f_i \in F} |P_f(\boldsymbol{x}_i) - f_i|,$$

$$\mathrm{RMAE}(X, F, \mathbf{I_f}) = \mathrm{RMAE}_{X,F}(\mathbf{I_f}) = \max_{\boldsymbol{x}_i \in X, f_i \in F} \frac{|P_f(\boldsymbol{x}_i) - f_i|}{f_i},$$

$$\mathrm{RRMSE}(X, F, \mathbf{I_f}) = \mathrm{RRMSE}_{X,F}(\mathbf{I_f}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\frac{P_f(\boldsymbol{x}_i) - f_i}{f_i} \right)^2},$$

where X and F are the given sets of data points and data values and $\mathbf{I_f} = (I_f(\boldsymbol{x}_1), \dots I_f(\boldsymbol{x}_N))$, with $N = |X|$.

The process begins by initializing the function g to be optimized, which is defined as the negative $\mathrm{MAE}_{X_{n_{val}}, F_{n_{val}}}(\cdot)$, representing the maximum absolute error between the known values $F_{n_{val}}$ and the approximation obtained by Algorithm 9 for the points $X_{n_{val}}$, given a specific value of θ and the search space $\mathcal{X}$. It is important to note that $X_{n_{val}}$ and $F_{n_{val}}$ are subsets of X_n, used specifically for the Bayesian optimization search. Subsequently, the algorithm proceeds until either the maximum number of iterations is reached or the error drops below a specified tolerance. For the initial *nstart* iterations, $\hat{\theta} = (\hat{\varepsilon}, \hat{\delta})$ is randomly sampled within the search domain $\mathcal{X}$. Afterwards, the chosen point $\hat{\theta}$ is the one that maximizes the acquisition function (3.5), evaluated over a randomly selected set. This acquisition function leverages the Gaussian process fitted in the previous iteration. Next, the subset $X_{n_j} \subset X_n$, which is contained within the corresponding subdomain Ω_j, is retrieved. This subset is then split into training and validation sets, and an approximant is fitted using Algorithm 9. During each iteration, the values of $\hat{\theta}$ and the error resulting from fitting the interpolant with the parameter $\hat{\theta}$ are stored in the vectors $\boldsymbol{\theta}$ and $\boldsymbol{g}$, respectively. A Gaussian process is then fitted to the pairs $(\boldsymbol{\theta}, \boldsymbol{g})$. In the final step, the optimal parameter θ^* is determined as the one that maximizes the vector $\boldsymbol{g}$.

The Algorithm 9, which represents the core of the scheme, solves the interpolation system and determines the approximations of the function on the points in $\bar{X}$. Algorithm 10 uses the values of ε and δ for each subdomain to compute local solutions, which are then combined to form a global solution. More specifically, the algorithm calculates the Shepard weights from (2.1) for each subdomain, and initializes the array for evaluating the approximation on $\bar{X}$ with zero values. It is important to note that the algorithms discussed in this section can be adapted to handle approximation problems by selecting a subset of X_n as the centres for the RBFs and solving a linear system of the form (1.13) in a least squares framework, as demonstrated in Chapter 4. This method is particularly advantageous in cases with large datasets or noisy data. In such situations, using an approximation rather than interpolation helps avoid fitting the noise. For more information, see [45].

5.2 Computational Analysis of Algorithms

In this section, we analyze the complexity of Algorithm 5 by breaking down its key components. Note that all logarithms are taken with base 2. Without specifying the shape parameter or subdomain radius, we will demonstrate that the total computational cost of

Algorithm 8 BO$(X, F, I, J, a, \xi, start, niter, DT_{X_n}, center, \tau)$

Input:

X: data points, F: data values, I: parameter search interval for ε, J: parameter search interval for δ, a: acquisition function, ξ: exploration-exploitation parameter, $nstart$: number of starting points, $niter$: number of Bayesian iterations, DT_{X_n}: distance tree of X_n, $center$: subdomain center, τ: tolerance.

Set $g \to -\text{MAE}_{X_{n_{j_{val}}}, F_{n_{j_{val}}}}(\cdot)$
$\mathcal{X} \to I \times J$
$\boldsymbol{g} \to (\cdot)$ (empty vector)
$\boldsymbol{\theta} \to (\cdot)$ (empty vector)
while $i \leq nstart + niter$ or $|\max(\boldsymbol{g})| > \tau$ **do**
  if $i \leq nstart$ **then**
    $\hat{\theta} \to$ random sample in $\mathcal{X}$ (note that $\hat{\theta} = (\hat{\varepsilon}, \hat{\delta})$)
  else
    Evaluate a on a set of random points in $\mathcal{X}$
    Select the point $\hat{\theta}$ that maximizes a
  end if
  $X_{n_j} \to$ retrieve the subset of X_n within distance $\hat{\delta}$ from $center$ using DT_{X_n}
  Split X_{n_j}, F_{n_j} in $X_{n_{j_{train}}}, X_{n_{j_{val}}}, F_{n_{j_{train}}}, F_{n_{j_{val}}}$
  $\mathbf{I}_{\boldsymbol{f}_{\hat{\theta}}} \to$ **RBF**$(X_{n_{j_{train}}}, F_{n_{j_{train}}}, X_{n_{j_{val}}}, \hat{\varepsilon})$ (call to **Algorithm** 9)
  $\boldsymbol{\theta} \to \boldsymbol{\theta} \cup \hat{\theta}$
  $\boldsymbol{g} \to \boldsymbol{g} \cup g(\mathbf{I}_{\boldsymbol{f}_{\hat{\theta}}}))$
  Fit the Gaussian process on $(\boldsymbol{\theta}, \boldsymbol{g})$
  $i \to i + 1$
end while
$\theta^* \to \text{argmax}\, \boldsymbol{g}$

Output:

θ^*: best parameters.

Algorithm 9 RBF$(X_n, F_n, \bar{X}, \varepsilon)$

Input:

X_n: data points, F_n: data values, $\bar{X}$: evaluation points, ε: shape parameter.

$\mathbf{I}_{\boldsymbol{f}} \to$ Solve the linear system of the form (1.13)

Output:

$\mathbf{I}_{\boldsymbol{f}}$: evaluation of the interpolated solution on $\bar{X}$.

Algorithm 10 PUM$(X_n, F_n, \bar{X}, centers, DT_{X_n}, w, \boldsymbol{\varepsilon}, \boldsymbol{\delta})$

Input:

X_n: data points, F_n: data values, $\bar{X}$: evaluation points, $centers$: subdomain centers, DT_{X_n}: distance tree of X_n, w: weight function, $\boldsymbol{\varepsilon}$: vector of shape parameters, $\boldsymbol{\delta}$: vector of subdomain radius.

$\mathbf{sw} \rightarrow$ retrieve Shepard weights from w
$\mathbf{I_f} \rightarrow$ 0-vector of length $|\bar{X}|$
$DT_{\bar{X}} \rightarrow$ distance tree of $\bar{X}$
for $j = 1 : |centers|$ **do**
  $X_{n_j} \rightarrow$ points of X_n at distance $\boldsymbol{\delta}[j]$ from $centers[j]$ using DT_{X_n}
  if $|X_{n_j}| \neq 0$ **then**
    $\bar{X}_{s_j} \rightarrow$ points of $\bar{X}$ at distance $\boldsymbol{\delta}[j]$ from $centers[j]$ using $DT_{\bar{X}}$
    if $\bar{X}_{s_j} \neq 0$ **then**
      $\mathbf{I'_{f_{s_j}}} \rightarrow \mathbf{RBF}(X_{n_j}, F_{n_j}, \bar{X}_{s_j}, \varepsilon_j)$
      $\mathbf{I_{f_{s_j}}} \rightarrow \mathbf{I_{f_{s_j}}} + \mathbf{I'_{f_{s_j}}} * \mathbf{sw_{s_j}}$
    end if
  end if
end for

Output:

$\mathbf{I_f}$: evaluation of the interpolated solution on $\bar{X}$

constructing a global interpolant is $\mathcal{O}(n^{\frac{2m+1}{m}} + n(nstart+niter)(n+n_j^3+(nstart+niter)^3))$, where $n = |X_n|$ represents the number of points, m is the dimension of the space, and n_j is the maximum number of points within a subdomain (with $n_j \ll n$). The terms *nstart* and *niter* refer to the initialization steps and the iterations of the Bayesian optimization, respectively. The complexity of constructing and searching within a k-d tree must also be considered, which is $\mathcal{O}(n\log(n))$ for construction and $\mathcal{O}(n)$ for searching. We utilize the k-d tree implementation provided by *scipy* [126], which, for a balanced dataset, builds a balanced tree using a median-based splitting approach in $\mathcal{O}(n\log(n))$. It is important to note that, while rare, the worst-case complexity for tree construction could reach $\mathcal{O}(n^2)$. As for search operations, three scenarios are worth highlighting: in the best case, for a balanced tree, the search cost is $\mathcal{O}(\log(n))$; in the average case, where the tree is reasonably balanced, the search cost becomes $\mathcal{O}(n+k)$, where k represents the number of points within the search radius. In the worst case, if the tree is unbalanced, the cost can rise to $\mathcal{O}(n)$.

FIND-MIN-RADIUS: Assume the initial radius for a subdomain is set to zero. In this scenario, the maximum number of augmenting steps within the *while* loop is limited by the diagonal length of the m-dimensional hypercube, scaled by the weight of the extent. This gives an upper bound of approximately $16d^{\frac{1}{m}} \simeq 8n^{\frac{1}{m}}$. Considering that the search itself has a cost of $\mathcal{O}(n)$, the total cost of the radius search in Algorithm 7 can be estimated as:

$$\begin{aligned}\mathcal{O}(d + d[n + 16d^{\frac{1}{m}}n]) &\simeq \mathcal{O}(d[n + 16d^{\frac{1}{m}}n]) \\ &\simeq \mathcal{O}(d^{\frac{m+1}{m}}n) \\ &\simeq \mathcal{O}(n^{\frac{2m+1}{m}}).\end{aligned}$$

RBF: Algorithm 9 simply involves the solution of a linear system. With an input of n nodes the computational expense is $\mathcal{O}(n^3)$.

PUM: Disregarding the computational cost of calculating the Shepard weights and the initialization steps, which scale linearly with n, the main computational effort comes from a for loop of length d. In each iteration, the loop includes three point searches, each costing $\mathcal{O}(n)$, a call to Algorithm 9 with variable input size, and the update of a vector of length n. Let $\tilde{n} = \max_j n_j = \max_j |X_j|$, representing the maximum number of points in any subdomain. With these considerations, the overall complexity of Algorithm 10 is:

$$\begin{aligned}\mathcal{O}(d[n + \tilde{n}^3]) &\simeq \mathcal{O}(n[n + \tilde{n}^3]) \\ &\simeq \mathcal{O}(n^2 + n\tilde{n}^3]).\end{aligned}$$

BO: Algorithm 8 consists of a *while* loop that runs for fewer than $nstart + niter$ iterations. During each iteration, the costs can be summarized as follows: an n-order computation to determine the next parameter for evaluation, a search involving 2 points, a cost split of n, a call to Algorithm 9 with a cost of $\tilde{n}^3$, and the fitting of the Gaussian process, which can incur a maximum cost of $(nstart + niter)^3$. Thus, the overall cost associated with the Bayesian optimization amounts to:

$$\mathcal{O}((nstart + niter)(n + \tilde{n} + (nstart + niter)^3)).$$

BO-PUM: In summary, by consolidating the earlier findings and considering that building the kd-trees necessitates computations on the order of $n\log(n)$, we can express the computational expense for Algorithm 5 as follows:

$$\begin{aligned}\mathcal{O}(n + n\log(n) + n^{\frac{2m+1}{m}} + d(start + niter)(n + \tilde{n}^3 + (nstart + niter)^3) + n^2 + n\tilde{n}^3)\\ \simeq \mathcal{O}(n^{\frac{2m+1}{m}} + n(start + niter)(n + \tilde{n}^3 + (nstart + niter)^3)\\ \simeq \mathcal{O}(n^{\frac{2m+1}{m}} + n^2(start + niter) + n\tilde{n}^3(start + niter) + n(nstart + niter)^4).\end{aligned}$$

5.3 Numerical Experiments

In this section, we will demonstrate how the algorithms outlined in Section 5.1 efficiently operate on test datasets. Before diving into specifics, it's important to note that all code was developed in Python 3.9, utilizing the *BayesianOptimization* library [88], which employs the Matérn 5/2 kernel as the default for the Gaussian process. We set the parameters $\xi = 0.15$ and $min_{pts} = 15$. To optimize the parameters (ε, δ), we focus on maximizing the Maximum Absolute Error (MAE) of the RBF interpolant, with the sign inverted, as Bayesian optimization (BO) is fundamentally a maximization process, as discussed in Section 3.2. We vary the number of points in the training set while keeping the test set constant at 1000 points. Throughout the BO process, once we identify the points in each subdomain, we further split them into sub-training and sub-validation sets to assess the training error. After determining the optimal parameter pairs for each subdomain, we train a PUM interpolant on the training set using the identified parameters. For each subdomain, the search space is defined as $\mathcal{X} = (0, 20] \times [\delta_{min}, 2\delta_{min}]$, where δ_{min} is the radius that ensures minimum density in the subdomain. BO conducts 5 random steps followed by up to 25 Bayesian steps within the search space. The iterative process concludes when the specified tolerance τ is achieved. Note that setting the tolerance to machine precision effectively compels the algorithm to complete all iterations. We conduct experiments on four different sizes of random datasets within the domain $\Omega = [0, 1]^2$, using three RBFs with varying smoothness, specifically:

$$\begin{aligned}\varphi_1(\varepsilon r) &= e^{-\varepsilon^2 r^2} && \text{(Gaussian } C^\infty\text{)},\\ \varphi_2(\varepsilon r) &= e^{-\varepsilon r}(1 + 3\varepsilon r + \varepsilon^2 r^2) && \text{(Matérn } C^4\text{)},\\ \varphi_3(\varepsilon r) &= (35\varepsilon^2 r^2 + 18\varepsilon r + 3)|1 - \varepsilon r|_+^6 && \text{(Wendland } C^4\text{)},\end{aligned}$$

and the following test functions [68, 97]:

$$\begin{aligned} f_1(x_1, x_2) = 0.75 \exp\left[-\frac{(9x_1-2)^2}{4} - \frac{(9x_2-2)^2}{4}\right] \\ + 0.75 \exp\left[-\frac{(9x_1-2)^2}{49} - \frac{9x_2+1}{10}\right] \\ + 0.5 \exp\left[-\frac{(9x_1-7)^2}{4} - \frac{(9x_2-3)^2}{4}\right] \\ - 0.2 \exp\left[-(9x_1-4)^2 - (9x_2-7)^2\right], \\ f_2(x_1, x_2) = 2\cos(10x_1)\sin(10x_2) + \sin(10x_1x_2). \end{aligned}$$

The results are presented in Tables 5.1, 5.2 and 5.3. Notably, as the number of points increases, the execution time for Bayesian optimization (BO) decreases. This reduction can be attributed to the higher density of the space with a larger number of points. Specifically, this increased density leads to more concentrated subdomains, resulting in improved accuracy and fewer BO iterations required to meet the tolerance level τ.

To evaluate the effectiveness of our predictions, we performed a Mann-Whitney U test [72]. In this test, the null hypothesis posits that the probability distribution of a randomly selected observation from one group is identical to that of a randomly selected observation from another group. The alternative hypothesis suggests that these distributions differ. For our analysis, the first sample consists of test data, while the second sample comprises the approximated values.

We employed the *mannwhitneyu* function from the *scipy* library [126] for all experiments, consistently resulting in a p-value exceeding 0.05. As this does not provide grounds to reject the null hypothesis, we infer that the samples likely come from the same distribution.

To emphasize the significance of the parameter search phase, we present in Table5.4 the results obtained using $\varepsilon = 10$, the midpoint of the search interval for ε, and $\delta = \sqrt{2/N}$, the minimum radius that guarantees the covering property. The results demonstrate that selecting parameters arbitrarily, without a clear strategy, leads to suboptimal outcomes.

Remark 5.3.1. As an example we compute the local condition numbers for each subdomain after optimizing on $N = 5000$ points using the function f_2 and the kernel φ_2. The results yield an average condition number of 5.15×10^{14} and a maximum condition number of 3.44×10^{17}.

All the tests reported in this and the following section are performed using Python 3.9.12 on a 1.2 GHz Quad-Core Intel Core i7 processor, 16 GB 3733 MHz LPDDR4X RAM MacBook Air (2020).

5.4 Applications to Real Data

In this subsection, we demonstrate the behavior of our PUM-BO framework on two real-world data examples. These examples highlight the algorithm's performance when measurements are taken at regular intervals, resembling grid points, and on contour lines, which simulate random measurements.

		Gaussian kernel (φ_1)		Matérn kernel (φ_2)	
N	τ	time (s)	MAE	time (s)	MAE
2000	1e-04	1.02e+01	8.16e-05	5.56e+01	2.15e-04
	1e-05	6.92e+01	1.00e-05	3.13e+02	1.66e-04
4000	1e-04	4.35e+00	2.68e-05	1.83e+01	6.81e-05
	1e-05	2.43e+01	5.50e-06	4.49e+02	4.36e-05
8000	1e-04	2.87e+00	9.14e-06	5.21e+00	3.28e-05
	1e-05	1.01e+01	5.49e-06	3.59e+02	3.00e-05
16000	1e-04	5.54e+00	1.25e-06	6.16e+00	3.59e-05
	1e-05	6.31e+00	1.07e-06	1.07e+02	2.07e-05

Table 5.1: Computational time and MAE using BO optimizer for Gaussian and Matérn kernels and different number N of random points in $\Omega = [0,1]^2$ using f_1 test function. Two tolerances τ for the training error are used.

		Gaussian kernel (φ_1)		Matérn kernel (φ_2)	
N	τ	time (s)	MAE	time (s)	MAE
2000	1e-04	3.79e+01	7.14e-05	3.98e+02	1.84e-02
	1e-05	2.39e+02	3.62e-04	3.97e+02	1.02e-02
4000	1e-04	1.35e+01	3.16e-05	6.73e+02	2.29e-03
	1e-05	1.32e+02	8.83e-06	7.55e+02	1.53e-03
8000	1e-04	6.18e+00	9.40e-05	6.43e+02	8.56e-04
	1e-05	5.47e+01	9.63e-06	1.46e+03	8.84e-04
16000	1e-04	5.65e+00	1.09e-05	1.36e+02	8.06e-05
	1e-05	1.87e+01	5.41e-06	2.78e+03	1.22e-04

Table 5.2: Computational time and MAE using BO optimizer for Gaussian and Matérn kernels and different number N of random points in $\Omega = [0,1]^2$ using f_2 test function. Two tolerances τ for the training error are used.

Tonga Trench Dataset: The Tonga Trench, located in the Pacific Ocean, reaches an extraordinary depth of 10,882 meters (35,702 feet) at its deepest point, Horizon Deep. This trench, along with a nearby volcanic island arc, forms a dynamic subduction zone where the Pacific Plate is being forced beneath the Tonga Plate, which is part of the northeastern edge of the Australian Plate. The trench extends north-northeast from the Kermadec Islands, situated northeast of New Zealand's North Island, before gradually bending westward near the Tonga Plate and transitioning into a region dominated by transform faults. The plates converge at an average rate of about 15 centimeters (6 inches) per year, but recent GPS data reveals localized convergence rates as high as 24 centimeters (10 inches) per year in the northern section, making it the fastest-moving tectonic boundary on Earth. These oceanic trenches are geologically significant because they play a key role in forming future continental crust and recycling materials back into the Earth's mantle. Within the Tonga Trench, molten rock from the mantle feeds island arc systems, while ocean floor sediments and fragments of oceanic crust accumulate over time. To the south, the Kermadec Trench is a continuation of the Tonga Trench, exhibiting unique characteristics that make it a compelling area for scientific research. In our example, we work with a dataset

		f_1		f_2	
N	τ	time (s)	MAE	time (s)	MAE
2000	1e-04	2.26e+02	1.35e-03	4.57e+02	1.07e-02
	1e-05	4.06e+02	1.57e-02	4.52e+02	3.11e-02
4000	1e-04	2.47e+02	3.49e-03	8.55e+02	2.81e-03
	1e-05	7.14e+02	7.10e-04	8.92e+02	3.95e-03
8000	1e-04	2.70e+02	4.15e-04	1.25e+03	1.27e-02
	1e-05	1.05e+03	1.23e-03	1.74e+03	2.41e-03
16000	1e-04	3.25e+02	1.59e-04	1.26e+03	8.06e-04
	1e-05	1.22e+03	1.15e-04	3.37e+03	7.25e-04

Table 5.3: Computational time and MAE using BO optimizer for Wendland kernel φ_3 and different number N of random points in $\Omega = [0,1]^2$ using f_1 and f_2 test functions. Two tolerances τ for the training error are used.

N	test function	kernel	time (s)	MAE
2000	f_1	φ_1	3.19e-05	1.18e+00
		φ_2	2.41e-05	1.18e+00
	f_2	φ_1	2.88e-05	1.94e+00
		φ_2	2.48e-05	1.94e+00
4000	f_1	φ_1	1.98e-05	9.86e-02
		φ_2	2.88e-05	9.57e-02
	f_2	φ_1	2.72e-05	3.50e-01
		φ_2	2.50e-05	3.38e-01
8000	f_1	φ_1	2.19e-05	7.65e-01
		φ_2	3.19e-05	7.64e-01
	f_2	φ_1	2.69e-05	2.65e+00
		φ_2	2.79e-05	2.65e+00
16000	f_1	φ_1	2.31e-05	4.84e-01
		φ_2	7.30e-05	4.84e-01
	f_2	φ_1	5.01e-05	2.33e+00
		φ_2	6.39e-05	2.33e+00

Table 5.4: Computational time and MAE for f_1 and f_2 test functions with Gaussian and Matérn kernels on different number N of random points in $\Omega = [0,1]^2$, applying the PUM without the Bayesian Optimization search. The shape parameter ε is set equal to 10, while the subdomain radius δ is set equal to $\sqrt{2/N}$.

containing 8113 points. We randomly split the data into a training set of 7000 points and a test set of 1113 points, ensuring no repetition. The indices are shuffled, with the first 7000 points allocated to the training set and the remaining points used for the test set (see Figure 5.1).

Franke's Glacier Dataset: This dataset, previously used in [39] for interpolating scattered data to fit a surface, comprises 8338 altitude measurements of a glacier. Unfortunately, we were unable to find background information regarding where these data were

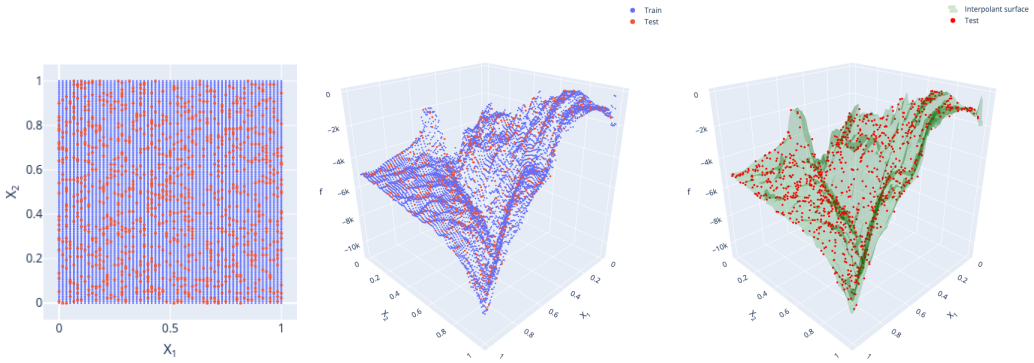

Figure 5.1: Tonga Trench dataset: plain projection (left), 3D view (center), interpolating surface constructed on a 100×100 point grid in $[0, 1]^2$.

	Gaussian kernel (φ_1)			Matérn kernel (φ_2)		
τ	time (s)	RMAE	RRMSE	time (s)	RMAE	RRMSE
1e-04	1.59e+03	6.99e-01	5.68e-02	1.34e+03	6.29e-01	5.45e-02
1e-05	1.57e+03	6.60e-01	6.14e-02	1.33e+03	6.16e-01	5.62e-02

Table 5.5: Computational time, RMAE and RRMSE using BO optimizer for Gaussian and Matérn kernels, φ_1 and φ_2, using Tonga dataset. Two tolerances τ for the training error are used.

collected or even the glacier's specific location. More details on this dataset can be accessed at https://search.r-project.org/CRAN/refmans/fields/html/glacier.html. The dataset is interesting, as the elevation measurements were taken along contour lines possibly digitized from a topographic map or a survey. In our example, we use the full dataset of 8338 points and randomly split it into a training set of 7000 points and a test set of 1338 points, without repetition. After shuffling the indices, the first 7000 points were assigned to the training set, while the remaining points were used for testing (see Figure 5.2).

As demonstrated by examples on both test and real datasets, when Bayesian Optimization (BO) is applied to simultaneously explore shape parameters and subdomain radii within the RBF-PUM framework, it efficiently steers the parameter selection toward optimal values, consistently achieving the predefined tolerance in all experiments. The key distinction lies in the search durations, which in some cases decrease as the number of points increases. This is because a larger dataset and denser point distribution within each subdomain allow the required tolerance to be met in fewer iterations, leading to shorter processing times.

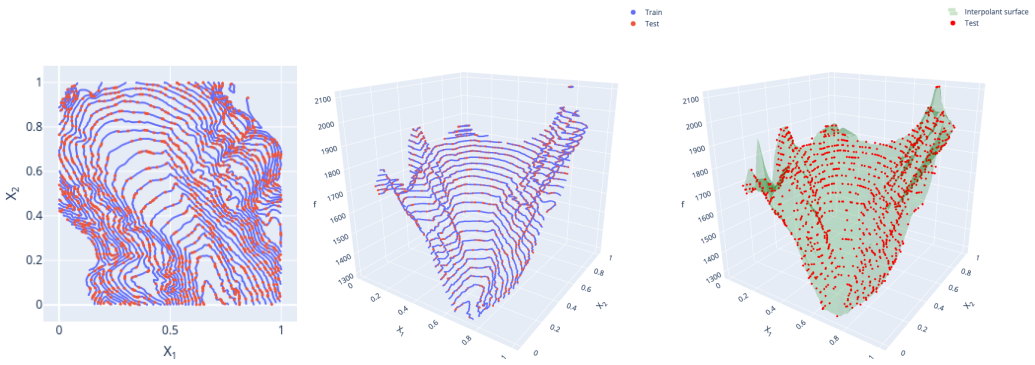

Figure 5.2: Franke's glacier dataset: plain projection (left), 3D view (center), interpolating surface constructed on a 100×100 point grid in $[0, 1]^2$.

	Gaussian kernel (φ_1)			Matérn kernel (φ_2)		
τ	time (s)	RMAE	RRMSE	time (s)	RMAE	RRMSE
1e-04	1.82e+03	9.25e-03	1.15e-03	1.62e+03	9.26e-03	8.74e-04
1e-05	1.83e+03	3.48e-02	1.49e-03	1.63e+03	9.33e-03	8.57e-04

Table 5.6: Computational time, RMAE and RRMSE using BO optimizer for Gaussian and Matérn kernels, φ_1 and φ_2, using glacier dataset. Two tolerances τ for the training error are used.

Chapter 6

Node-Bound Communities for Partition of Unity Interpolation on Graphs

Graph signal processing has become a widely used approach for exploring graph signals, which has led to extensive research on mathematical techniques like filtering, compression, noise reduction, sampling, and decomposition, as discussed in various studies [90, 124]. Additionally, graphs frequently arise in diverse applications, such as modeling social networks or mapping traffic patterns [27]. However, their typically intricate structure demands fast and effective tools for thorough analysis. A key feature of real-world graphs is the organization of vertices into communities, where dense connections exist within the same community, while links between different communities are comparatively sparse. Identifying these communities is crucial in fields like sociology, biology, and computer science, where systems are often represented as graphs, yet this remains a complex and unresolved challenge [50]. Partition of unity methods (PUMs) allow for performing tasks such as reconstructing signals from samples, classifying nodes, or applying localized filtering to specific graph segments [27]. This approach enables the recomposition of the overall signal by merging these localized parts.

In the context of scattered data approximation, combining radial basis functions (RBFs) with PUMs leads to significantly sparser system matrices for collocation and interpolation tasks, which in turn greatly decreases computation time [32, 23, 43]. In this chapter our focus will be on approximation techniques that employ generalized shifts of a graph basis function (GBF); for more details, refer to [41] and [42].

We introduce a more adaptable approach for selecting graph partitions compared to the method in [27]. Specifically, by treating graph communities as potential partitions for the PUM, we present a technique that automatically determines an optimal number of communities or subdomains based on the modularity and the structure of the graph. Notably, this approach does not require the desired number of subdomains as input for the GBF-PUM algorithm, representing an innovation in this field. Our findings are supported by a series of numerical experiments conducted to validate this new algorithm on both synthetic and real-world datasets.

The remainder of this chapter revisits work initially published in [30], co-authored by the author of this thesis. Section 6.1 reviews foundational concepts in graph theory, while Section 6.2 introduces the problem of signal approximation on graphs using positive definite GBFs. Section 6.3 provides a detailed description of our algorithm, including pseudocode, and explains its integration into the GBF-PUM schema proposed in [27]. Section 6.4 discusses the algorithm's complexity analysis, and Section 6.6 presents numerical examples in both basic and real-world graph contexts.

6.1 Graph theory

A *graph* G is a couple (V, E), where $V(G) \neq \emptyset$ and $E(G)$ are the set of *vertices* or *nodes* and the set of *edges* or *links* of the graph G respectively. The set of edges E is constituted by couple of vertices in V. The order of the graph G is the cardinality of its vertex set $|V(G)|$. When we want to emphasize the dependence of G from its nodes and edges sets we will write $G = (V(G), E(G))$.

Definition 6.1.1. Let G a graph:

- A *loop* is an edge of the form (v, v);
- *Multiple edges* are two or more edges having the same end points.

If G has neither loops nor multiple edges, G is called *simple*.

Definition 6.1.2. Let G be a graph. A *path* is an alternate sequence without repetition of nodes and edges that join 2 different nodes. If, for any couple of vertices in G, there exists a joining path, G is connected.

In what follows, we consider only simple, connected graphs. Readers interested in more detailed arguments, definitions, and notation not explicitly provided in this chapter are referred to [15] and [87].

Definition 6.1.3. Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be two graphs. If $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, we say that H is a *subgraph* of G. Any collections $H_1, \ldots, H_p$ of subgraphs of G such that they are pairwise disjoint and their union forms a cover of $V(G)$ is called *partition* of G and the subgraphs $H_1, \ldots, H_p$ are called *communities*.

Definition 6.1.4. Let G be a graph and $V(G)$ be the set of its vertices. For any vertex $u \in V(G)$:

- the *neighbourhood* of u is the set $N(u) = \{v \in V(G) | (u, v) \in E(G)\}$;
- the *degree* of u, denoted as $deg_G(u)$, is $|N(u)|$ (the number of edges incident to u).

A convenient way to describe the structure of a graph G given its set of vertices $V(G) = \{u_1, \ldots, u_n\}$ is to exploit matrices as follows.

Definition 6.1.5. Let $G = (V(G), E(G))$ be a graph with $|V(G)| = n$. We define the *adjacency matrix* of the graph G as a squared matrix A of order n such that

$$A_{ij} = \begin{cases} 1 & \text{if} \quad (u_i, u_j) \in E(G) \\ 0 & \text{otherwise} \end{cases}$$

Definition 6.1.6. Let G be a graph with $|V(G)| = n$. Let A be its adjacency matrix, and D the diagonal squared matrix of order n with entries $D_{ii} = deg_G(u_i)$.
The squared matrix L of order n, defined as $L = D - A$, is the *Laplacian matrix* of the graph G.

Centrality measures are effective tools for ranking nodes by their importance within a graph. Various definitions of centrality exist, each calculated differently to highlight distinct features of nodes. In this context, we base our community search on the following centrality measure.

Definition 6.1.7. *Katz centrality* assesses the relative influence of a vertex within a graph by counting both its immediate neighbors and all other vertices that are linked to it through these neighbors. However, connections to more distant neighbors are weighted less by an attenuation factor, α, typically set to 0.5. The Katz centrality for a vertex v_i is defined as:

$$C_{Katz}(u_i) = \sum_{k=1}^{\infty} \sum_{j=1}^{n} \alpha^k (A^k)_{ij}.$$

In Section 6.3 we will need the notion of *cut* in a graph used in the algorithms to discern communities.

Definition 6.1.8. Let G be a graph, C_1, C_2 subgraphs of G such that $V(C_1) \bigcap V(C_2) = \emptyset$ and $V(C_1) \bigcup V(C_2) = V(G)$. The partition of $V(G)$ generating C_1, C_2 is said *cut* $C = (C_1, C_2)$ and the set of edges having a vertex in C_1 and the other one in C_2 is said the *cut-set* of C. Given $u, v \in V(G)$, an (u, v)-cut is a cut $C = (C_1, C_2)$ such that $u \in C_1$ and $v \in C_2$.

Definition 6.1.9. In an undirected, unweighted graph, the *capacity* of a cut C is defined as the number of edges in the cut-set of C. If the graph has weighted edges, the capacity is instead the sum of the edge weights in the cut-set. A cut with the smallest weight, or with a weight equal to any other minimal cut, is referred to as the *minimum cut*.

Definition 6.1.10. Let G be a graph and $u, v \in V(G)$. The *Jaccard similarity index* $J(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$ is a measure of the similarity between the vertices u and v.
Given two communities U, V of G the Jaccard similarity index for U and V is

$$J(U, V) = \operatorname*{avg}_{u \in U, v \in V} \{J(u, v)\}.$$

Note that the Jaccard similarity index is limited between 0 and 1.

Let us now introduce a measure of the strength of a community subdivision of a graph.

Definition 6.1.11. Let G be a graph, A its adjacency matrix and $C = \{C_1, \ldots, C_n\}$ a division in community of G. The modularity of G related to $C = \{C_1, \ldots, C_n\}$ is defined as

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j),$$

where m is the number of edges, k_i is the degree of u_i and $\delta(C_i, C_j)$ is 1 if u_i and u_j are in the same community, zero otherwise. For unweighted and undirected graphs the modularity falls in the interval $[-1/2,1]$ with positive values when the number of connections inside the communities is greater than expected in a randomly generated graph.

Let us note that graphs with high modularity have dense connections between the vertices within communities but sparse connections between vertices in different communities.

6.2 Positive definite GBFs for signal processing on graphs

When the number of measurements is limited, Graph-Based Functions (GBFs) provide simple and effective tools for signal interpolation and approximation on graphs. The key benefit of using GBFs is an improvement in the accuracy of results. The related theory is closely linked to that of meshfree approximation with positive definite Radial Basis Functions (RBFs) in Euclidean space (see [111] and [133] for further details).

Definition 6.2.1. Let G be a graph with $|V(G)| = n$, a function $x : V(G) \to \mathbb{R}$ defined on the vertices of G is called *graph signal* and the n-dimensional vector space of all graph signals will be denoted by $\mathcal{L}(G)$.

Using the Laplacian matrix L, a Fourier transform can be defined on the graph G because $L = UM_\lambda U^T$. Here $M_\lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ represents the diagonal matrix whose elements are the eigenvalues of L arranged in ascending order. The columns of the orthonormal matrix U correspond to the normalized eigenvectors of L for each respective eigenvalue. The collection $\hat{G} = \{u_1, \ldots, u_n\}$, made up of these ordered eigenvectors, provides an orthonormal basis for the space $\mathcal{L}(G)$ and is called the *spectrum* of G.

Definition 6.2.2. Let G be a graph and $L = UM_\lambda U^T$ its Laplacian. Given a signal $x \in \mathcal{L}(G)$ on the graph G, the Fourier transform of x on the graph G is $\hat{x} = U^T x$. The inverse Fourier transform is defined by $x = U\hat{x}$.

We now turn our attention to kernels $K : V \times V \to \mathbb{R}$ defined on G such that, for all $u, v \in V$, it holds that $K(u, v) = K(v, u)$. Such kernels allow us to define a linear operator $\mathbf{K} : \mathcal{L}(G) \to \mathcal{L}(G)$ which acts on $x \in \mathcal{L}(G)$ as follows:

$$\mathbf{K}x(v_i) = \sum_{j=1}^{n} K(v_i, v_j) x(v_j).$$

Thus, $\mathbf{K}$ can be represented by a symmetric matrix of size $n \times n$ as:

$$K = \begin{bmatrix} K(v_1, v_1) & K(v_1, v_2) & \dots & K(v_1, v_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(v_n, v_1) & K(v_n, v_2) & \dots & K(v_n, v_n) \end{bmatrix}.$$

We define a symmetric kernel function K as *positive definite* if the associated matrix K is strictly positive definite, meaning $x^T K x \geq 0$ for all $x \in \mathbb{R}^n, x \neq 0$. Note that any positive definite kernel K induces an inner product on the space $\mathcal{L}(G)$ given by $\langle x, y \rangle_K = y^T K x$, along with a norm $\|x\|_K$ for every $x, y \in \mathcal{L}(G)$. This inner product space is called the *native space* $\mathcal{N}_K$, which also forms a reproducing kernel Hilbert space; see [4]. To compute a GBF approximation $x_\star$ for a signal x based on N samples $x(w_i)$, where $i = 1, \dots, N$, we aim to find $x_\star$ by solving the following regularized least squares (RLS) problem:

$$x_\star = \text{argmin}_{y \in \mathcal{N}_\mathbf{K}} \left\{ \frac{1}{N} \sum_{i=1}^{N} |x(w_i) - y(w_i)|^2 + \gamma \|y\|^2_{\mathbf{K}_f} \right\}. \tag{6.1}$$

Where the first term in Equation (6.1) is the data fidelity term which ensures that the values $x_\star(w_i)$ are close to the given values $x(w_i)$ on the sampling set W. The parameter $\gamma > 0$ is referred to as the regularization parameter.

Numerical examples presented in Section 6.6 rely on polyharmonic splines on graphs, which are an example of a positive definite GBF based on the kernel:

$$K_{f_{(\epsilon I_n + L)^{-s}}} = (\epsilon I_n + L)^{-s} = \sum_{k=1}^{n} \frac{1}{(\epsilon + \lambda_k)^s} u_k u_k^T,$$

where $u_1, \dots, u_n$ are the eigenvectors of the Laplacian L. If λ_1 is the largest eigenvalue of L, this kernel is positive definite for $\epsilon > -\lambda_1$ and $s > 0$ (see [92] and [129]). Hence values of $\epsilon = -\lambda_1 + 1$ and $s = 1$ are set for the experiments.

6.3 Overlapping communities detection

Signal interpolation on graphs using PUM involves creating overlapping subdomains, each containing at least one vertex that serves as an interpolation node. In this section, we outline a method for identifying these subdomains for PUM on graphs, leveraging the structural topology of the graph itself.

Community detection is a prominent area in graph theory, with various methodologies available; for an in-depth review, see [87]. Our approach follows a divisive strategy, where the centrality of interpolation nodes is used as the primary criterion for splitting, allowing communities to overlap. The community identification process is broken down into five interconnected functions. Notably, when the adjacency matrix A of a graph G is used as algorithm input, it inherently captures all information on the nodes $V(G)$ and edges $E(G)$. The primary function, Algorithm 12, accepts the adjacency matrix A of G and the set of interpolation (or sample) nodes W as input. It then iteratively attempts to divide communities until either the desired number of interpolation nodes is achieved or further division no longer enhances modularity. During each division attempt, Algorithm 13 generates a new candidate sub-partition for each community, along with its modularity

by splitting the specified community. The sub-partition with the highest modularity, if it exceeds the initial modularity, replaces the previous partition. Subsequently, small communities are merged with larger ones using Algorithm 15, and finally, Algorithm 16 is employed to expand communities, creating the necessary overlap for GBF-PUM.

Algorithm 12 community_detection$(A, W, r, dmax, dmin)$

Input:
A: adjacency matrix of the graph G, W: interpolation vertices, r: ratio of neighbours of a vertex inside the community over the total number, $dmax$: distance of max augmentation, $dmin$: distance of min augmentation.

$partition \to \{V(G)\}$
$Q' \to -1$
$Q \to modularity(partition)$
while $|partition| <= |W|$ & $Q' < Q$ **do**
  $Q' \to Q$
  for $i \leq |partition|$ **do**
    $partition_i, Q_i \to find\_partition(A, partition, partition(i), W)$
    if $Q_i > Q$ **then**
      $partition \to partition_i$
      $Q \to Q_i$
    end if
  end for
end while
$partition \to join\_communities(A, partition)$
$expanded\_partition \to expand\_communities(A, partition, r, dmax, dmin)$

Output:
$partition$: partition of V, $expanded\_partition$: expanded partition with overlap of V.

Algorithm 13 is designed to generate a candidate sub-partition. Its inputs include the adjacency matrix A of the graph G, a *partition* of $V(G)$, the community to be split, and the set of sample nodes W. The process begins by calling Algorithm 14 to obtain two sub-communities resulting from the division. It then creates the sub-partition and calculates its modularity, providing these as output.

The central step of the entire process is handled by Algorithm 14. Given a connected graph (either G or one of its communities) and the set of interpolation nodes W as input, it first calculates the Katz centrality for each sample vertex (see Definition 6.1.7). Next, it identifies the two vertices s and v with the highest centrality values. The capacity (see Definition 6.1.9) of all edges with s or v as the first endpoint, and where the other endpoint is neither v, s, nor one of their respective neighbours, is set to infinity. All remaining edges are assigned a capacity of 1. Finally, a minimum (s, v)-cut (see Definition 6.1.8) is performed, yielding the two sub-communities that are then returned as output.

Algorithm 15 is responsible for merging smaller communities into the most similar larger ones. With inputs A and the *partition*, it begins by classifying communities as either small,

Algorithm 13 find_partition$(A, partition, community, W)$

Input:

A: adjacency matrix of the graph G, *partition*: partition of $V(G)$, *community*: a community of G, W: interpolation vertices.

C_1, $C_2 \rightarrow split\_net(A, community, W)$
$partition' \rightarrow (partition \setminus community) \cup \{C_1, C_2\}$
$Q' \rightarrow$ modularity of $partition'$

Output:

$partition'$: the sub-partition, Q': modularity of $partition'$.

Algorithm 14 split_net(A, G, W)

Input:

A: adjacency matrix of the graph G, G: connected graph G, W: interpolation vertices.

Evaluate the Katz centrality for each vertex in W
Select s and v, the vertices with high centrality
for $\{x, y\} \in E(G)$ **do**
  $capacity(\{x, y\}) \rightarrow 1$
end for
for $u \in N(s)$ **do**
  if $u \notin N(v) \cup \{v\}$ **then**
    $capacity(\{s, u\}) \rightarrow \infty$
  end if
end for
for $u \in N(v)$ **do**
  if $u \notin N(s) \cup \{s\}$ **then**
    $capacity(\{v, u\}) \rightarrow \infty$
  end if
end for
$C_1, C_2 \rightarrow$ Compute the value and the node partition of a minimum $(s, v) - cut$

Output:

C_1: split community, C_2: split community, $partition'$: new partition, Q': modularity of $partition'$.

if they contain fewer than 2% of the total vertices, or large. For each small community, the algorithm calculates the Jaccard similarity (see Definition 6.1.10) between the small community and each large community in the graph. The small community is then merged with the large community to which it is most similar.

Algorithm 15 join_communities$(A, partition)$

Input:
A: adjacency matrix of the graph G, *partition*: partition of $V(G)$.

Divide *partition* in *small_communities* and *big_communities* (a community is small if contains less than the 2% of vertices)
for *community* in *small_communities* **do**
find the highest Jacard index between the *community* and the *big_communities*
merge the communities with the highest Jacard index
update *partition*
end for
Output:
partition: partition of V without small communities.

The final step in identifying communities is carried out by Algorithm 16. Given inputs A, *partition*, r, *dmax*, and *dmin*, the algorithm operates as follows: for each vertex in a community, it calculates the ratio of neighbours within the community to the total number of neighbours. If this ratio falls below r, the community is expanded to include all vertices within a distance of *dmax* from the selected vertex; otherwise, it is expanded to include all vertices within a distance of *dmin*. The output provides the overlapping communities required for GBF-PUM interpolation (Algorithm 17).

6.4 Computational complexity of Algorithm 12

To assess the computational complexity of Algorithm 12, we must analyze the complexity of the auxiliary algorithms it invokes. Note that logarithms are taken with base 2. In detail, within Algorithm 14 (*split_net*), the minimum-cut operation requires $O(|C_i||E(C_i)| + |C_i|^2 \log(|C_i|))$ for each community C_i. Thus, the total computation time for one iteration is:

$$\sum_{i=1}^{|C|} O(|C_i||E(C_i)| + |C_i|^2 \log(|C_i|)) \leq O(|V||E| + |V|^2 \log(|V|)).$$

Additionally, calculating Katz centrality for the vertices in G has complexity $O(|V| + |E|)$, assuming the power iteration method is applied. Therefore, the complexity of Algorithm 14 aligns with that of the minimum-cut operation. The complexity of computing modularity for each community must also be considered. By tracking the modularity changes Q throughout the algorithm, this step has a complexity bounded by $O(|V| + |E|)$. Moreover, computing the Jaccard index between communities has a worst-case complexity of $O(|V|^2)$, given the scenario where $|C| = |V|$. All remaining operations are performed in linear time. Consequently, the complexities of modularity evaluation and Jaccard index calculation do

Algorithm 16 expand_communities$(A, partition, r, dmax, dmin)$

Input:
A: adjacency matrix of the graph G, *partition*: partition of $V(G)$, r: ratio of neighbours of a vertex inside the community over the total number, $dmax$: distance of max augmentation, $dmin$: distance of min augmentation.

for $i \leq |partition|$ **do**
  for v in $partition(i)$ **do**
    $N(v) \rightarrow$ neighbours of v
    $N(v)_{in} \rightarrow$ neighbours of v inside *community*
    if $|N(v)_{in}| < r * |N(v)|$ **then**
      add to $partition(i)$ all the vertices within distance of $dmax$ from v
    else
      add to $partition(i)$ all the vertices within distance of $dmin$ from v
    end if
  end for
end for

Output:
partition: partition of V with expanded communities.

not impact the overall time complexity, as they are both faster than Algorithm 14. In the worst-case scenario where the number of iterations reaches $|V|$, the total time complexity becomes:

$$O(|V|^2|E| + |V|^3 \log(|V|)) \approx O(|V|^3 \log(|V|)),$$

which is polynomial.

6.5 GBF-PUM approximation on graphs

Partition of unity is a commonly utilized tool in mesh-free approximation problems, as it significantly reduces computational costs when addressing related subproblems [45]. This approach has been adapted for graph signal interpolation in [35] and [27] and can be applied to issues like social network analysis or traffic mapping due to their graph-based structures. After identifying the overlapping communities within a graph, constructing a partition of unity requires a set of functions $\varphi^{(j)} \in \mathcal{L}(G)$, $j = 1, \ldots, |C|$, such that:

- $\mathrm{supp}(\varphi^{(\mathrm{j})}) \subseteq \mathrm{V_j}$;
- $\varphi^{(j)} \geq 0$;
- $\sum_{j=1}^{|C|} \varphi^{(j)}(v) = 1$, $\forall v \in V$.

In particular, we choose the characteristic function

$$\varphi^{(j)}(v) = \mathbb{1}_{V_j}(v) = \begin{cases} 1 & \text{if } v \in V_j, \\ 0 & \text{if } v \notin V_j, \end{cases}$$

to obtain the partition of unity as follows:

$$\varphi^{(j)}(v) = \frac{\mathbb{1}_{V_j}(v)}{|j \in \{1, \ldots, |C| : v \in V_j\}|}.$$

A key step in this process is calculating the local approximants $x_\star^{(j)}$ on the subgraph G_j corresponding to the j-th community. This is achieved using a GBF approximation scheme, as detailed in Subsection 6.2. Once these local approximants are computed, the global GBF-PUM approximation across the entire graph G is given by:

$$x_\star(v) = \sum_{j=1}^{|C|} \varphi^{(j)}(v) x_\star^{(j)}(v).$$

It is important to note that overlapping subdomains are essential for achieving an effective PUM. Additional information on the error associated with the global approximation can be found in [27]. Algorithm 17 outlines the steps required to obtain the global interpolant. Unlike previous pseudocode, Algorithm 17 is not described in detail, as all the steps, except community detection, are well-established in the literature. Further information is available in [27, Section 4].

Algorithm 17 GBF-PUM Approximation on Graphs$(A, W, r, dmax, dmin)$

Input:

A: adjacency matrix of the graph G, W: interpolation vertices, r: ratio of neighbours of a vertex inside the community over the total number, $dmax$: distance of max augmentation, $dmin$: distance of min augmentation.

Find the overlapping communities C applying community detection (Algorithm 12)
Construct a partition of unity subordinate to the communities in C
For all subgraphs C_i in C calculate the local Laplacian L^c
For all subgraphs C_i in C construct the local GBF kernel
For all subgraphs C_i in C calculate the local GBF approximant
Create a global GBF-PUM approximation from the local ones

Output:

$x_\star$: GBF-PUM approximant of the signal.

6.6 Numerical examples

In this section, we demonstrate the subdivision capabilities of the algorithm by examining Zachary's Karate Club graph [137]. We then present numerical results obtained by applying our framework to the Minnesota Road graph [99], where a test function serves as the signal on its vertices, as well as to the real-world freight transport dataset from Brussels [57]. For testing, we set the parameters $r = 0.75$, $dmax = 6$, and $dmin = 4$ as inputs for Algorithm 1.

All tests were performed on the high-performance computing infrastructure *MathHPC*, a virtual cloud server within the *HPC4AI* (High-Performance Computing for Artificial Intelligence) structure: https://www.dipmatematica.unito.it/do/progetti.pl/View?doc=Laboratori_di_ricerca.html).

6.6.1 Zachary's Karate Club graph

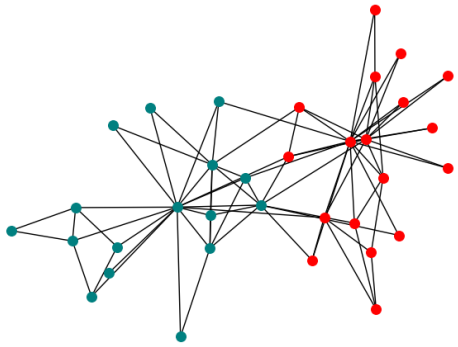

Figure 6.1: The Zachary's Karate Club graph two communities.

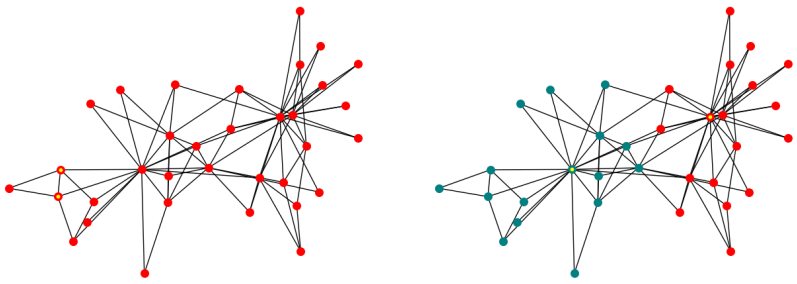

Figure 6.2: On the left it is shown how a bad selection of the sample nodes leads to a non-subdivision of the network. On the right, when the two sample nodes are chosen as the instructor and the administrator of the karate club the algorithm finds the ground truth two communities. The sample nodes are dotted in yellow.

Zachary's Karate Club graph represents a social network with 34 vertices and 78 edges, modeling interactions within a university karate club as described in [137]. During the study, a dispute emerged between the club's administrator and instructor, leading the club to divide into two groups. Half of the members formed a new club around the instructor, while the rest either sought a new instructor or discontinued karate. These original communities are shown in Figure 6.1. Let $W = \{u_1, u_2\}$ represent the set of sampling nodes for the community detection algorithm, assuming u_1 and u_2 are adjacent. Running Algorithm 12 with this choice of nodes will not yield a further subdivision, as no modularity gain can be achieved by separating the two sample nodes. Alternatively, selecting u_1 and u_2 as the nodes associated with the instructor and administrator leads our algorithm to perform a single division, which results in the familiar two-community structure (see Figure 6.2).

6.6.2 Minnesota Road graph

The Minnesota Road graph is a network with 2642 vertices and 3304 edges, where vertices represent intersections and edges correspond to roads within the State of Minnesota [118]. To apply the presented algorithms, a test function derived from the Laplacian matrix, as introduced in [35] and [27], has been modified and used as a signal across the vertex set. In Figure 6.3 the communities are shown, while in Table 6.1 numerical results are summarized. It is important to note that the number of communities identified by the divisive Algorithm 12 is solely determined by the centrality values of the sampled vertices. Table 6.1 presents the number of communities detected, the Relative-Max-Absolute-Error (RMAE), the Relative-Root-Mean-Squared-Error (RRMSE), and the computational time as the number of interpolation vertices varies.

W	Communities	RMAE	RRMSE	time (s)
400	11	3.509e-01	3.208681e-02	1.425e+02
800	9	9.276e-02	5.880306e-03	1.879e+02
1200	11	1.977e-02	1.444769e-03	1.587e+02
1600	10	1.517e-02	6.787503e-04	1.586e+02
2000	9	5.078e-03	2.122356e-04	1.690e+02

Table 6.1: A summarisation of the numerical results for increasing number of sampling nodes in the Minnesota Road graph. The number of communities, RMAE, RRMSE, and computational time are also shown.

6.6.3 Brussel Freight Transport network

The Brussels Freight Transport network is a graph with 4540 vertices and 14,946 edges, representing the city's road network [57]. For 3451 vertices, traffic flow data are recorded every 15 minutes. In our numerical test, we use a single time-point flow measurement as the signal for the sample vertices. The related real-world task involves reconstructing missing flow values using GBF-PUM interpolation, while, for testing, we focused on the largest connected component of vertices with known flow data. A sample of identified communities is shown in Figure 6.4, and numerical results are presented in Table 6.2.

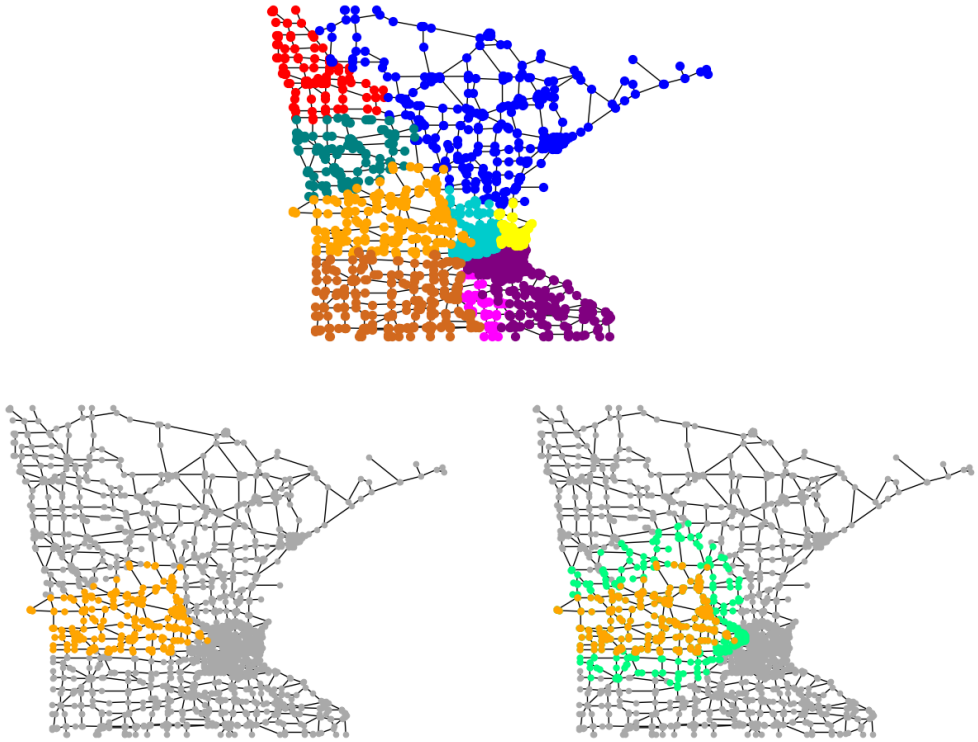

Figure 6.3: On top, the partition of the vertices in 9 different disjoint subdomains is shown. It is obtained using 800 interpolation nodes. On bottom, a subdomain is highlighted to show how it is expanded when creating the overlap (left to right).

W	Communities	RMAE	RRMSE	time (s)
400	15	1.500e+01	6.881386e-02	2.767e+02
800	7	9.173e-01	6.087108e-02	2.752e+02
1200	15	8.015e-01	4.673573e-02	2.768e+02
1600	9	6.286e-01	3.925505e-02	2.253e+02
2000	17	5.845e-01	3.011343e-02	2.168e+02
2400	16	6.538e-01	2.435230e-02	2.166e+02
2800	12	3.589e-01	1.770847e-02	1.974e+02
3200	6	1.732e-01	7.304001e-03	1.629e+02

Table 6.2: A summarisation of the numerical results for increasing number of sampling nodes in the Brussel Freight Transport network. The number of communities, RMAE, RRMSE and computational time are also shown.

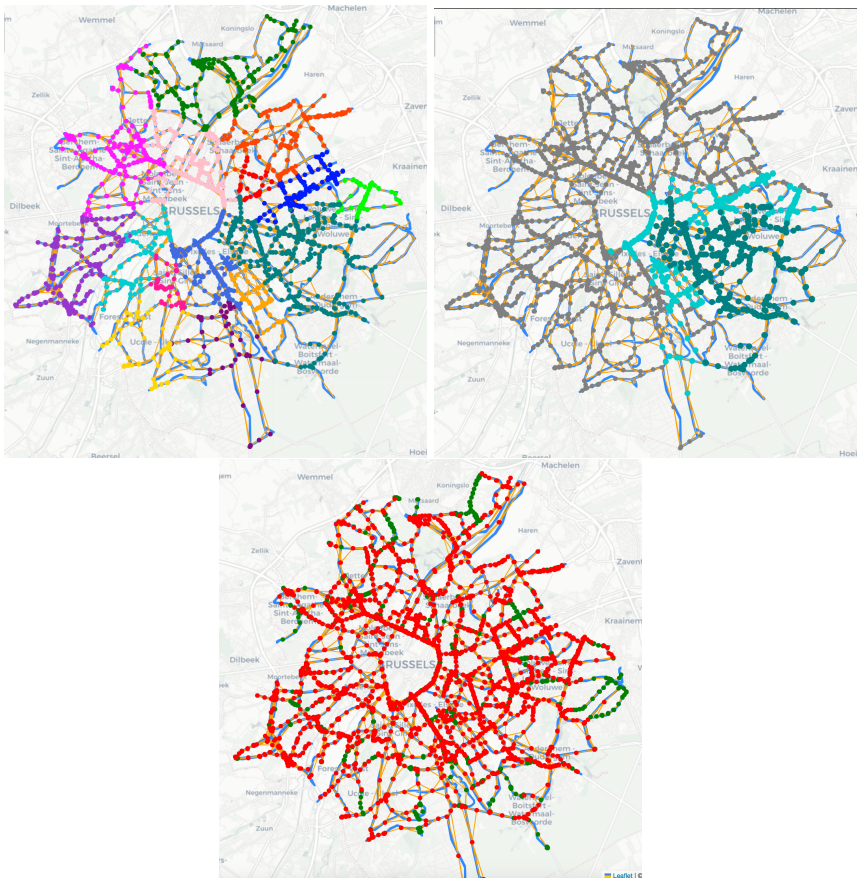

Figure 6.4: On top the division in communities of the Brussel Freight Transport network using 1200 sample nodes and the expansion of one of them is shown (left to right). On bottom the sample set of measures is highlighted in red, whereas the unknown values to predict are in green.

6.7 Discussion on results

Based on the numerical results presented in Table 6.1 and Table 6.2, we observe that as the number of sampled nodes increases, so does the accuracy of the GBF-PUM method. In real dataset applications, this trend is somewhat less pronounced yet remains noticeable due to the inherent variability in the data. Notably, the proposed approach does not require specifying the number of subdomains as an input, allowing for an automatic determination of a reasonable subdomain count for PUMs. This challenge is extensively studied beyond graph and network contexts, including in sparse data interpolation, approximation for PDE solutions [67], and within stochastic modeling frameworks [33]. An automated procedure for community augmentation could also be explored.

Bibliography

[1] R. A. Adams, *Sobolev Spaces*, Academic Press, New York, 1975.

[2] G. Allasia, R. Cavoretto, A. De Rossi, Hermite-Birkhoff interpolation on scattered data on the sphere and other manifolds. Appl. Math. Comput. 318, (2018) 35–50.

[3] L. Alzubaidi, J. Zhang, A.J. Humaidi, Y. Duan, J. Santamaría, M.A. Fadhel, L. Farhan, Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions, Journal of Big Data 8 (2021) 1–74.

[4] N. Aronszajn, Theory of reproducing kernels, Trans. Amer. Math. Soc. 68 (1950), 337–404.

[5] I. Babuška, J.M. Melenk, The partition of unity method, Internat. J. Numer. Methods Engrg. 40(4) (1997), 727–758.

[6] K. Ball, Eigenvalues of Euclidean distance matrices, *J. Approx. Theory* **68** (1992), 74–82.

[7] K. Ball, N. Sivakumar, J. D. Ward, On the sensitivity of radial basis interpolation to minimal data separation distance, *Constr. Approx.* **8** (1992), 401–426.

[8] M. D. Buhmann, New developments in the theory of radial basis function interpolation, in: K. Jetter, F. Utreras (Eds.), *Multivariate Approximation: from CAGD to Wavelets*, World Scientific, Singapore, 1993, pp. 35–75.

[9] M.D. Buhmann. Radial Basis Functions: Theory and Implementation. Cambridge Monogr. Appl. Comput. Math., vol. 12, Cambridge Univ. Press, Cambridge, 2003.

[10] M.E. Biancolini Fast Radial Basis Functions for Engineering Applications, Springer Cham, 2018.

[11] B. Biazar, M. Hosami, An interval for the shape parameter in radial basis function approximation. Appl. Math. Comput. 315 (2017) 131–149.

[12] C.M. Bishop, Neural networks and their applications, Rev. Sci. Instrum. 65 (1994) 1803–1832.

[13] S. Bochner, *Vorlesungen über Fouriersche Integrale*, Akademische Verlagsgesellschaft, Leipzig, 1932.

[14] S. Bochner, Monotone funktionen, Stieltjes integrale und harmonische analyse, *Math. Ann.* **108** (1933), 378–410.

[15] J.A. Bondy, U.S.R. Murty, Graph Theory, Springer Series: Graduate Texts in Mathematics. Springer London, 2008.

[16] M. Bozzini, L. Lenarduzzi, M. Rossini, Polyharmonic splines: An approximation method for noisy scattered data of extra-large size, Appl. Math. Comput. 216, 317–331 (2010)

[17] E. Brochu, V.M. Cora, N. De Freitas, A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010, arXiv:1012.2599

[18] M.D. Buhmann, Radial Basis Functions: Theory and Implementation, Cambridge Monogr. Appl. Comput. Math., vol. 12, Cambridge Univ. Press, Cambridge (2003)

[19] R. E. Carlson, T. A. Foley, Interpolation of track data with radial basis methods, *Comput. Math. Appl.* **12** (1992), 27–34.

[20] R. E. Carlson, B. K. Natarajan, Sparse approximate multiquadric interpolation, *Comput. Math. Appl.* **27** (1994), 99–108.

[21] R. Cavoretto, A. De Rossi. A trivariate interpolation algorithm using a cube-partition searching procedure. *SIAM J. Sci. Comput.*, 37:A1891–A1908, 2015.

[22] R. Cavoretto, Adaptive radial basis function partition of unity interpolation: A bivariate algorithm for unstructured data, J. Sci. Comput. 87 (2021) 41.

[23] R. Cavoretto, A. De Rossi, Error indicators and refinement strategies for solving Poisson problems through a RBF partition of unity collocation scheme, Appl. Math. Comput. 369 (2020), 124824.

[24] R. Cavoretto, A. De Rossi, F. Dell'Accio, F. Di Tommaso, Fast computation of triangular Shepard interpolants, J. Comput. Appl. Math. 354 (2019) 457–470.

[25] R. Cavoretto, A. De Rossi, F. Dell'Accio, F. Di Tommaso. An efficient trivariate algorithm for tetrahedral Shepard interpolation. *J. Sci. Comput.*, 82:57, 2020.

[26] R. Cavoretto, A. De Rossi, W. Erb, GBFPUM - A MATLAB Package for Partition of Unity Based Signal Interpolation and Approximation on Graphs, Dolomites Res. Notes Approx. 15 (2022), 25–34.

[27] R. Cavoretto, A. De Rossi, W. Erb, Partition of unity methods for signal processing on graphs, J. Fourier Anal. Appl. 27 (2021), 66.

[28] R. Cavoretto, A. De Rossi, S. Lancellotti, Bayesian approach for radial kernel parameter tuning, J. Comput. Appl. Math. 441, 115716 (2024)

[29] R. Cavoretto, A. De Rossi, S. Lancellotti, E. Perracchione, Software implementation of the partition of unity method, Dolomites Res. Notes Approx. 15 (2022) 35–46.

[30] R. Cavoretto, A. De Rossi, S. Lancellotti, F. Romaniello, Node-bound communities for partition of unity interpolation on graphs, Appl. Math. Comput. 467 (2024) 128502.

[31] R. Cavoretto, A. De Rossi, S. Lancellotti, F. Romaniello, Parameter tuning in the radial kernel-based partition of unity method by Bayesian optimization, J. Comput. Appl. Math. 451 (2024) 116108.

[32] R. Cavoretto, A. De Rossi, M.S. Mukhametzhanov, Ya.D. Sergeyev, On the search of the shape parameter in radial basis functions using univariate global optimization methods, J. Global Optim. 79 (2021) 305–327.

[33] R. Cavoretto, A. De Rossi, E. Perracchione, Learning with partition of unity-based kriging estimators, Appl. Math. Comput. 448 (2023), 127938.

[34] R. Cavoretto, A. De Rossi, E. Perracchione. Optimal selection of local approximants in RBF-PU interpolation. *J. Sci. Comput.*, 74:1–22, 2018.

[35] R. Cavoretto, A. De Rossi, A. Sommariva, M. Vianello, RBFCUB: A numerical package for near-optimal meshless cubature on general polygons, Appl. Math. Lett. 125 (2022) 107704.

[36] R. Charles, K. Harris, J. Millman, J.S. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T.E. Oliphant. Array programming with NumPy. *Nature* 585:357–362, 2020.

[37] C.-S. Chen, A. Noorizadegan, D.L. Young, C.S. Chen, On the selection of a better radial basis function and its shape parameter in interpolation problems, Appl. Math. Comput. 442 (2023) 127713.

[38] P. C. Curtis Jr., n-parameter families and best approximation, *Pacific J. Math.* **9** (1959), 1013–1027.

[39] O. Davydov, F. Zeilfelder, Scattered data fitting by direct extension of local polynomials to bivariate splines, Adv. Comput. Math. 21 (2004), 223–271.

[40] T. A. Driscoll, B. Fornberg, Interpolation in the limit of increasingly flat radial basis functions, *Comput. Math. Appl.* **43** (2002), 413–422.

[41] W. Erb, Graph Signal Interpolation with Positive Definite Graph Basis Functions, Appl. Comput. Harmon. Anal. 60 (2022), 368–395.

[42] W. Erb, Semi-Supervised Learning on Graphs with Feature-Augmented Graph Basis Functions. arXiv:2003.07646 (2020).

[43] G.E. Fasshauer, Meshfree Approximation Methods with MATLAB, World Scientific, Singapore, 2007.

[44] G. E. Fasshauer, On smoothing for multilevel approximation with radial basis functions, in: C. K. Chui, L. L. Schumaker (Eds.), *Approximation Theory IX: Computational Aspects*, Vanderbilt Univ. Press, 1999, pp. 55–62.

[45] G.E. Fasshauer, M.J. McCourt, Kernel-based Approximation Methods Using MATLAB, World Scientific, Singapore, 2015.

[46] G.E. Fasshauer, J.G. Zhang, On choosing "optimal" shape parameters for RBF approximation, Numer. Algorithms 45 (2007) 345–368.

[47] W. Feller, *An Introduction to Probability Theory and its Application*, vol. 2, Wiley & Sons, New York, 1971.

[48] T.A. Foley, Near optimal parameter selection for multiquadric interpolation, J. Appl. Sci. Comput. 1, 54–69 (1994)

[49] B. Fornberg, G. Wright, Stable computation of multiquadrics interpolants for all values of the shape parameter, Comput. Math. Appl. 47 (2004) 497–523.

[50] S. Fortunato, Community detection in graphs, Phys. Report 486 (2010), 75–174.

[51] E. Francomano, M. Paliaga, Highlighting numerical insights of an efficient SPH method, Appl. Math. Comput. 339 (2018) 899–915.

[52] R. Franke, Scattered data interpolation: tests of some methods, Math. Comp. 48 (1982) 181—200.

[53] R. Franke, H. Hagen. Least squares surface approximation using multiquadrics and parametric domain distorsion. *Comput. Aided Geom. Design*, 16:177–196, 1999.

[54] L. Gatteschi, *Funzioni Speciali*, Unione Tipografico – Editrice Torinese, Torino, 1973.

[55] A. Golbabai, E. Mohebianfar, H. Rabiei, On the new variable shape parameter strategies for radial basis functions, Comput. Appl. Math. 34, 691–704 (2015)

[56] M.A. Golberg, C.S. Chen, S.R. Karur, Improved multiquadric approximation for partial differential equations, Eng. Anal. Bound. Elem. 18 (1997) 9–17.

[57] T. Guns, S. Hadavi, C. Macharis, S. Verlinde, W. Verbekem Monitoring Urban-Freight Transport Based on GPS Trajectories of Heavy-Goods Vehicles, IEEE Transactions on Intelligent Transportation Systems 20(10) (2019), 3747–3758.

[58] J.H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numer. Math.*, 2:84–90, 1960.

[59] K. Hamacher, On stochastic global optimization of one-dimensional functions, Physica A: Statistical Mechanics and its Applications 354, 547–557 (2005)

[60] R.L. Hardy, Multiquadric equations of topography and other irregular surfaces, J. Geophys. Res. 76 (1971) 1905–1915.

[61] A. Iske, Scattered data approximation by positive definite kernel functions, Rend. Sem. Mat. Univ. Pol. Torino 69 (2011) 217–246.

[62] D.E. Johnson, Introduction to Filter Theory, Prentice Hall Inc., New Jersey (1976)

[63] D.R. Jones, M. Schonlau, W.J. Welch, Efficient Global Optimization of Expensive Black-Box Functions, J. Global Optim. 13 (1998) 455–492.

[64] H.Y.F. Lam, Analog and Digital Filters-Design and Realization, Prentice Hall Inc., New Jersey (1979)

[65] E. Larsson, B. Fornberg, Theoretical and computational aspects of multivariate interpolation with increasingly flat radial basis functions, Comput. Math. Appl. 49 (2005) 103–130.

[66] E. Larsson, R. Schaback, Scaling of radial basis functions, IMA Journal of Numerical Analysis, 2023, drad035, https://doi.org/10.1093/imanum/drad035

[67] E. Larsson, V. Shcherbakov, A. Heryudono, A least squares radial basis function partition of unity method for solving PDEs, SIAM J. Sci. Comput. 39 (2017), A2538–A2563.

[68] D. Lazzaro, L.B. Montefusco, Radial basis functions for the multivariate interpolation of large scattered data sets, J. Comput. Appl. Math. 140, 521–536 (2002)

[69] L. Ling, F. Marchetti, A stochastic extended Rippa's algorithm for LpOCV, Appl. Math. Letters 129 (2022) 107955.

[70] D. Lizotte, Practical Bayesian Optimization, PhD thesis, University of Alberta, Edmonton, Alberta, Canada, 2008.

[71] J. C. Mairhuber, On Haar's theorem concerning Chebychev approximation problems having unique solutions, *Proc. Amer. Math. Soc.* **7** (1956), 609–615.

[72] H.B. Mann, D.R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, Ann. Math. Stat. 18 (1947) 50–60.

[73] F. Marchetti, The extension of Rippa's algorithm beyond LOOCV, Appl. Math. Letters 120 (2021) 107262.

[74] F. Marchetti, E. Perracchione. Local-to-Global Support Vector Machines (LGSVMs). *Pattern Recognit.*, 132:108920, 2022.

[75] G. Matheron, *Les Variables Régionalisées et leur Estimation*, Masson, Paris, 1965.

[76] M. Mathias, Über positive Fourier-Integrale, *Math. Z.* **16** (1923), 103–125.

[77] MATLAB, Natick, Massachusetts, The Mathworks, Inc. R2019b.

[78] J.M. Melenk, I. Babuˇska, The partition of unity finite element method: Basic theory and applications, Comput. Meth. Appl. Mech. Eng. 139 (1996), pp. 289–314.

[79] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, Philadelphia, PA, 2000.

[80] C. A. Micchelli, Interpolation of scattered data: distance matrices and conditionally positive definite functions, *Constr. Approx.* **2** (1986), 11–22.

[81] D. Mirzaei. The direct radial basis function partition of unity (D-RBF-PU) method for solving PDEs. *SIAM J. Sci. Comput.* 43:A54–A83, 2021.

[82] J. Mockus, V. Tiesis, A. Zilinskas, The application of Bayesian methods for seeking the extremum, Towards Global Optimization 2 (1978) 117–129.

[83] A. Molinaro, Y.D. Sergeyev, Finding the minimal root of an equation with the multiextremal and nondifferentiable left-hand part, Numer. Alg. 28, 255–272 (2001)

[84] F. J. Narcowich, N. Silvakumar, J. D. Ward, On condititon numbers associated with radial-function interpolation, *J. Math. Anal. Appl.* **186** (1994), 457–485.

[85] F. J. Narcowich, J. D. Ward, Norms and inverses and condition numbers for matrices associated with scattered data, *J. Approx. Theory* **64** (1991), 69–94.

[86] F. J. Narcowich, J. D. Ward, Norms and inverses for matrices associated with scattered data, in: P.-J. Laurent, A. Le Méhauté, L. L. Schumaker (Eds.), *Curves and Surfaces*, Academic Press, New York, 1991, pp. 341–348.

[87] M. Newman, Networks: An Introduction, OUP Oxford, 2010, 784 pp.

[88] F. Nogueira, Bayesian optimization: Open source constrained global optimization tool for Python, https://github.com/fmfn/BayesianOptimization

[89] A. Noorizadegan, C.-S. Chen, R. Cavoretto, A. De Rossi, Efficient truncated randomized SVD for mesh-free kernel methods, Comput. Math. Appl. 164, 12–20 (2024)

[90] A. Ortega, P. Frossard, J. Kovačević, J.M.F. Moura, P. Vandergheynst, Graph signal processing: overview, challenges, and applications, Proc. IEEE 106(5) (2018), 808–828.

[91] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas,A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[92] I.Z. Pesenson, Variational splines and Paley-Wiener spaces on combinatorial graphs, Constr. Approx. 29(1) (2009), 1–21.

[93] J.D. Pinter, Global Optimization in Action – Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications, Kluwer Academic Publishers, Dordrecht (1996)

[94] R. B. Platte, T. A. Driscoll, Polynomials and potential theory for Gaussian radial basis function interpolation, *SIAM J. Numer. Anal.*, **43** (2005), 750–766.

[95] R Core Team (2020). R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria.

[96] C.E. Rasmussen, C. Williams, Gaussian Processes for Machine Learning, MIT Press, 2006.

[97] R.J. Renka, R. Brown, Algorithm 792: Accuracy test of ACM algorithms for interpolation of scattered data in the plane, ACM Trans. Math. Software 25 (1999) 78–94.

[98] S. Rippa, An algorithm for selecting a good value for the parameter c in radial basis function interpolation, Adv. Comput. Math. 11 (1999) 193–210.

[99] R.A. Rossi, N.K. Ahmed, The Network Data Repository with Interactive Graph Analytics and Visualization, in Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, *http://networkrepository.com*.

[100] A. Safdari-Vaighani, A. Heryudono, E. Larsson. A radial basis function partition of unity collocation method for convection-diffusion equations arising in financial applications. *J. Sci. Comput.*, 64:341–367, 2015.

[101] B. Schölkopf, A.J. Smola. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA, USA, 2002.

[102] D. Shepard, A two-dimensional interpolation function for irregu- larly spaced data, in: Proceedings of 23-rd National Conference, Bran- don/Systems Press, Princeton, 1968, pp. 517–524.

[103] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, N. de Freitas, Taking the human out of the loop: A review of Bayesian optimization, Proceedings of the IEEE, 104-1 (2016) 148–175.

[104] V. Shcherbakov, E. Larsson, Radial basis function partition of unity methods for pricing vanilla basket options, Comput. Math. Appl. 71 (2016), pp. 185–200.

[105] D. Shepard. A two-dimensional interpolation function for irregularly spaced data. In: Proceedings of 23-rd National Conference, Brandon/Systems Press, Princeton, 517–524, 1968.

[106] R. Schaback, Error estimates and condition numbers for radial basis function interpolation, *Adv. Comput. Math.* **3** (1995), 251–264.

[107] R. Schaback, Lower bound for norms of inverses of interpolation matrices for radial basis functions, *J. Approx. Theory* **79** (1994), 287–306.

[108] R. Schaback, Multivariate interpolation and approximation by translates of a basis function, in: C. Chui, L. L. Schumaker (Eds.), *Approximation Theory VIII: Approximation and Interpolation*, World Scientific, Singapore, 1995, pp. 491–514.

[109] R. Schaback, Small errors imply large evaluation instabilities, Adv. Comput. Math. 49 (2023) 25.

[110] R. Schaback, Stability of radial basis function interpolants, in: C. K. Chui, L. L. Schumaker, J. Stöckler (Eds.), *Approximation Theory X*, Vanderbilt Univ. Press, Nashville, TN, 2002, pp. 433–440.

[111] R. Schaback, H. Wendland, Approximation by Positive Definite Kernels, in Advanced Problems in Constructive Approximation, Birkhäuser Verlag, Basel, 2003, pp. 203–222

[112] R. Schaback, H. Wendland Kernel techniques: from machine learning to meshless methods, Acta Numer. 15, 543–639 (2006)

[113] R. Schaback, Z. Wu, Operators on radial functions, *J. Comput. Appl. Math.* **73** (1996), 257–270.

[114] M. Scheuerer, An alternative procedure for selecting a good value for the parameter c in RBF-interpolation, Adv. Comput. Math. 34 (2011) 105–126.

[115] I. J. Schoenberg, Metric spaces and completely monotone functions, *Ann. of Math.* **39** (1938), 811–841.

[116] I. J. Schoenberg, Selected papers, vol. 1, C. de Boor (Ed.), with contributions by P. Erdos, J.-L. Goffin, S. Cambanis, D. Richards, R. Askey, S. Ruscheweyh, Birkhäuser, Boston, MA, 1988.

[117] I. J. Schoenberg, Selected papers, vol. 2, with contributions by C. de Boor, S. Karlin, G. G. Lorentz, Birkhäuser, Boston, MA, 1988.

[118] D.I. Shuman, B. Ricaud, P. Vandergheynst, Vertex-frequency analysis on graphs, Appl. Comput. Harm. Anal. 40(2) (2016), 260–291.

[119] Y.D. Sergeyev, D.E. Kvasov, F.M.H. Khalaf, A one-dimensional local tuning algorithm for solving GO problems with partially defined constraints, Optim. Lett. 1(1), 85–99 (1995)

[120] Y.D. Sergeyev, M.S. Mukhametzhanov, D.E. Kvasov, D. Lera, Derivative-free local tuning and local improvement techniques embedded in the univariate global optimization, J. Optim. Theory Appl. 171, 186–208 (2016)

[121] D. Shepard, A two-dimensional interpolation function for irregularly-spaced data, in Proceedings of the 23rd National Conference ACM (1968) 517–523.

[122] I. H. Sneddon, *The Use of Integral Transforms*, McGraw-Hill, New York, 1972.

[123] J. Snoek, H. Larochelle, R.P. Adams, Practical Bayesian Optimization of Machine Learning Algorithms, Advances in Neural Information Processing Systems 25 (2012) 2960–2968.

[124] L. Stanković, L. Daković, E. Sejdić, Introduction to Graph Signal Processing, in Vertex-Frequency Analysis of Graph Signals, Springer, 2019, pp. 3–108.

[125] J. Stewart, Positive definite function and generalizations, an historical survey, *Rocky Mountain J. Math.* **6** (1976), 409–434.

[126] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272.

[127] C.J. Trahan, R.W. Wyatt, Radial basis function interpolation in the quantum trajectory method: optimization of the multi-quadric shape parameter, J. Comput. Phys. 185 (2003) 27–49.

[128] M. Uddin, On the selection of a good value of shape parameter in solving time-dependent partial differential equations using RBF approximation method, Appl. Math. Model. 38 (2014) 135–144.

[129] J.P. Ward, F.J. Narcowich, J.L. Ward, Interpolating splines on graphs for data science applications, Appl. Comput. Harm. Anal. 49(2) (2020), 540–557.

[130] H. Wendland, Error estimates for interpolation by compactly supported radial basis functions of minimal degree, *J. Approx. Theory* **93** (1998), 258–272.

[131] H. Wendland, Fast evaluation of radial basis functions: methods based on partition of unity, in Approximation theory X: wavelets, splines and applications, Vanderbilt University Press, Nashville, (2002) 473–483.

[132] H. Wendland, Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree, *Adv. Comput. Math.* **4** (1995), 389–396.

[133] H. Wendland, Scattered Data Approximation, Cambridge Monogr. Appl. Comput. Math., vol. 17, Cambridge Univ. Press, Cambridge, 2005.

[134] D. V. Widder, *The Laplace Transform*, Princeton Univ. Press, Princeton, 1941.

[135] R. E. Williamson, Multiply monotone functions and their Laplace transform, *Duke Math. J.* **23** (1956), 189–207.

[136] Z. Wu, R. Schaback, Local error estimates for radial basis function interpolation of scattered data, *IMA J. Numer. Anal.* **13** (1993), 13–27.

[137] W.W. Zachary, An Information Flow Model for Conflict and Fission in Small Groups, Jour. of Anthropological Research 33(4) (1977), 452–473.