Journal of Scientific Computing (2021) 87:41
https://doi.org/10.1007/s10915-021-01432-z

Adaptive Radial Basis Function Partition of Unity Interpolation: A Bivariate Algorithm for Unstructured Data

Roberto Cavoretto[1]

Received: 28 October 2020 / Revised: 16 January 2021 / Accepted: 9 February 2021 /
Published online: 18 March 2021

Abstract
In this article we present a new adaptive algorithm for solving 2D interpolation problems of large scattered data sets through the radial basis function partition of unity method. Unlike other time-consuming schemes this adaptive method is able to efficiently deal with scattered data points with highly varying density in the domain. This target is obtained by decomposing the underlying domain in subdomains of variable size so as to guarantee a suitable number of points within each of them. The localization of such points is done by means of an efficient search procedure that depends on a partition of the domain in square cells. For each subdomain the adaptive process identifies a predefined neighborhood consisting of one or more levels of neighboring cells, which allows us to quickly find all the subdomain points. The algorithm is further devised for an optimal selection of the local shape parameters associated with radial basis function interpolants via leave-one-out cross validation and maximum likelihood estimation techniques. Numerical experiments show good performance of this adaptive algorithm on some test examples with different data distributions. The efficacy of our interpolation scheme is also pointed out by solving real world applications.

Keywords Meshless approximation · Partition of unity methods · Radial basis functions · Scattered data interpolation · Adaptive algorithms

Mathematics Subject Classification 65D05 · 65D15 · 41A05

1 Introduction

In kernel based approximation radial basis function (RBF) methods are effective meshfree techniques, which can be implemented to numerically solve various types of science and engineering problems. Over the last years the use of RBF methods has gained much attention in several interdisciplinary fields. Indeed, while many traditional numerical methods such as finite differences, finite elements or finite volumes have troubles with high-dimensional prob-

✉ Roberto Cavoretto
roberto.cavoretto@unito.it

1 Department of Mathematics “Giuseppe Peano”, University of Torino, via Carlo Alberto 10, 10123 Torino, Italy

lems, meshfree methods can often handle changes in the geometry of the domain of interest (e.g., free surfaces, moving particles and large deformations) better. Moreover, independence from a mesh is a great advantage since mesh generation is one of the most time-consuming parts of any mesh-based numerical simulation (see e.g. [16,17,20]). All these issues deserve to be considered not only when one has to model (systems of) partial differential equations (PDEs), but also when a multivariate problem of scattered data fitting needs to be faced and solved appropriately. This problem is particularly relevant in several situations in which surface reconstruction involves unstructured large data sets, requiring in some way the construction of adaptive interpolation or approximation methods. Approximating scattered data of high complexity arises in various areas of applied sciences, ranging from scanner acquisitions to geographic benchmarks, as well as for industrial and medical purposes where the processing of large random data configurations is usually carried out. In such cases scattered data can have significantly different distributions, e.g. data with highly varying density or data with voids, which demand adaptive algorithms (see e.g. [8,13]).

In this article we present a new adaptive algorithm for solving 2D interpolation problems of large scattered data sets. In doing that, though some adaptive strategies have been developed, for example, in [15,38], a global RBF interpolation method is not definitely suitable for our purposes, since it seldom turns out to be usable in practice. Indeed, a global scheme is characterized by a big interpolation matrix, and this results in a two-fold issue: first, a severe ill-conditioning of the matrix with a consequent high level of instability; second, a high computational cost when such a matrix has to be inverted (and the method applied in its entirety). For this reason, in this work, we focus on a local RBF method, such as the radial basis function partition of unity method (RBF-PUM), which allows us to decompose a big problem into several small subproblems. This interpolation method relies on a decomposition of the domain into several subdomains forming a cover of it, and constructing a local RBF interpolant on each subdomain. The original idea of PUM comes from the context of PDEs [3, 27], but later it also gained much popularity in the field of numerical approximation [10,11,13] and, more in general, in various areas of applied mathematics and scientific computing (see e.g. [4,5,12,19,22,23,25,28]). However, although the RBF-PUM has some specific features that makes it particularly suitable for processing large scattered data sets, the problem of interpolating very irregularly distributed data points has been addressed only partly in [13]. In fact, the numerical method in [13] is accurate, but at the same time it is also quite costly from the computational point of view. Here we therefore propose a new adaptive scheme that allows us to define subdomains of variable size so as to guarantee in any case a suitable number of points within each of them. Such points are localized by considering a cell based structure, which preliminary partitions the underlying domain and its points in a suitable number of cells, thus enabling the use of an efficient search procedure. After completing this phase, the algorithm identifies for each subdomain optimal values of the local RBF shape parameters via *leave-one-out cross validation* (LOOCV) or *maximum likelihood estimation* (MLE). Both techniques have a statistical background and can be combined with efficient optimization routines to quickly achieve reliable predictions of the RBF shape parameters (see [16,30,31]). In our extensive numerical experiments we show the performance of this new adaptive algorithm, also comparing our results with non-adaptive and adaptive—but existing or standard routine based—algorithms. Finally, the efficacy of our scheme is also pointed out by solving interpolation problems with data sets coming from real world applications.

The paper is organized as follows. In Sect. 2 we give a brief overview on the basic notions regarding the RBF based interpolation. In Sect. 3 we focus on the construction of the RBF-PUM selecting local RBF shape parameters via either LOOCV or MLE. Section 4 is devoted to describe our adaptive algorithm, then discussing computational issues and providing a complexity analysis. In Sect. 5 we present several numerical results designed on test examples in order to illustrate the performance of our interpolation algorithm. Section 6 shows some applications to real world data sets. Finally, Sect. 7 deals with conclusions and future work.

2 Preliminaries on RBF Based Interpolation

In this section we give a brief overview on the basic notions of RBF methods, which are powerful and flexible tools for scattered data interpolation. To have more details on the theoretical background, we refer the reader to [9,16,36].

Given a domain $\Omega \subseteq \mathbb{R}^s$, a set $X_N = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \subseteq \Omega$ of N distinct *data points* or *nodes* and the corresponding *data* or *function values* $f(\boldsymbol{x}_i) \in \mathbb{R}$, $i = 1, \dots, N$, obtained by possibly sampling any (unknown) function $f : \Omega \to \mathbb{R}$, we want to find a function $s : \Omega \to \mathbb{R}$ that satisfies the interpolation conditions

$$s(\boldsymbol{x}_i) = f(\boldsymbol{x}_i), \quad i = 1, \dots, N. \tag{1}$$

We express a RBF interpolant $s : \Omega \to \mathbb{R}$ as a linear combination of RBFs, i.e.,

$$s(\boldsymbol{x}) = \sum_{i=1}^{N} c_i \phi_\varepsilon(||\boldsymbol{x} - \boldsymbol{x}_i||_2), \quad \boldsymbol{x} \in \Omega,$$

where c_i, $i = 1, \dots, N$, are unknown real coefficients, $||\cdot||_2$ denotes the Euclidean norm, and $\phi : \mathbb{R}_{\geq 0} \to \mathbb{R}$ is a strictly positive definite (SPD) RBF depending on a *shape parameter* $\varepsilon > 0$ such that

$$\phi_\varepsilon(||\boldsymbol{x} - \boldsymbol{y}||_2) = \phi(\varepsilon||\boldsymbol{x} - \boldsymbol{y}||_2), \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \Omega.$$

In the following, for the sake of simplicity, we refer to ϕ_ε as ϕ. In Table 1 we report a list of some SPD RBFs together with their degrees of smoothness. Note that Gaussian, Inverse MultiQuadric and Matérn functions are globally supported and SPD in $\mathbb{R}^s$ for any s, while Wendland functions are compactly supported—with support $[0, 1/\varepsilon]$—and SPD in $\mathbb{R}^s$ for $s \leq 3$ [36].

Since ϕ is a SPD function, the matrix $\mathsf{A} = (\mathsf{A}_{ki})$ with the entries $\mathsf{A}_{ki} = \phi(||\boldsymbol{x}_k - \boldsymbol{x}_i||_2)$, $k, i = 1, \dots, N$, is positive definite for all possible sets of nodes. In this case, the coefficients c_k are uniquely determined by enforcing the interpolation conditions in (1) and can be obtained by solving the symmetric linear system

$$\mathsf{A}\boldsymbol{c} = \boldsymbol{f}, \tag{2}$$

where $\boldsymbol{c} = (c_1, \dots, c_N)^T$, and $\boldsymbol{f} = (f_1, \dots, f_N)^T$. Therefore, the interpolation problem is well-posed and, hence, its solution exists uniquely [17].

Moreover, for any SPD RBF ϕ we can define a symmetric and SPD kernel $\Phi : \Omega \times \Omega \to \mathbb{R}$, i.e.,

$$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \phi(||\boldsymbol{x} - \boldsymbol{y}||_2), \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \Omega.$$

For the kernel Φ there exists the so-called *native space*, that is a Hilbert space $\mathcal{N}_\Phi(\Omega)$ with inner product $(\cdot,\cdot)_{\mathcal{N}_\Phi(\Omega)}$ in which the kernel Φ is reproducing, i.e., for any $f \in \mathcal{N}_\Phi(\Omega)$ we have the identity $f(\boldsymbol{x}) = (f, \Phi(\cdot, \boldsymbol{x}))_{\mathcal{N}_\Phi(\Omega)}$, for $\boldsymbol{x} \in \Omega$. Thus, the space $H_\Phi(\Omega) = \text{span}\{\Phi(\cdot, \boldsymbol{x}), \boldsymbol{x} \in \Omega\}$, equipped with the bilinear form $(\cdot,\cdot)_{H_\Phi(\Omega)}$, is an inner product space with reproducing kernel Φ. The native space $\mathcal{N}_\Phi(\Omega)$ of the kernel Φ is then defined as the

Table 1 Some examples of popular SPD RBFs

RBF	$\phi_\varepsilon(r)$
Gaussian C^∞ (GA)	$\exp(-\varepsilon^2 r^2)$
Inverse MultiQuadric C^∞ (IMQ)	$(1+\varepsilon^2 r^2)^{-1/2}$
Matérn C^6 (M6)	$\exp(-\varepsilon r)(\varepsilon^3 r^3 + 6\varepsilon^2 r^2 + 15\varepsilon r + 15)$
Matérn C^4 (M4)	$\exp(-\varepsilon r)(\varepsilon^2 r^2 + 3\varepsilon r + 3)$
Matérn C^2 (M2)	$\exp(-\varepsilon r)(\varepsilon r + 1)$
Wendland C^6 (W6)	$\max(1-\varepsilon r, 0)^8\,(32\varepsilon^3 r^3 + 25\varepsilon^2 r^2 + 8\varepsilon r + 1)$
Wendland C^4 (W4)	$\max(1-\varepsilon r, 0)^6\,(35\varepsilon^2 r^2 + 18\varepsilon r + 3)$
Wendland C^2 (W2)	$\max(1-\varepsilon r, 0)^4\,(4\varepsilon r + 1)$

completion of $H_\Phi(\Omega)$ with respect to the norm $||\cdot||_{H_\Phi(\Omega)}$, i.e. $||f||_{H_\Phi(\Omega)} = ||f||_{\mathcal{N}_\Phi(\Omega)}$, for any $f \in H_\Phi(\Omega)$ (see [17,36]).

3 RBF-PUM Based Interpolation

In this section we focus on the construction of the method, describing the LOOCV and MLE based approaches that can be used to select optimal values of the RBF shape parameters. Moreover, we present the main theoretical results concerning the RBF-PUM interpolation.

3.1 Construction of RBF-PUM Interpolants

Let $\Omega \subseteq \mathbb{R}^s$ be an open and bounded domain, and let $\{\Omega_j\}_{j=1}^d$ be an open and bounded cover of Ω that fulfills some mild overlap condition among the subdomains Ω_j. Indeed, the subdomains Ω_j need to form a cover of the domain such that $\bigcup_{j=1}^d \Omega_j \supseteq \Omega$. Moreover, a given point $\boldsymbol{x} \in \Omega$ must belong at most to K (independent of d) overlapping subdomains. A typical example of partition of unity (PU) subdomains using scattered data points in $\mathbb{R}^2$ is shown in Fig. 1; in this case, for the sake of clarity, we show circular subdomains of fixed radius.

Given the subdomains Ω_j, we consider a partition of unity $\{w_j\}_{j=1}^d$ subordinated to the cover $\{\Omega_j\}_{j=1}^d$ such that

$$\sum_{j=1}^d w_j(\boldsymbol{x}) = 1, \quad \boldsymbol{x} \in \Omega,$$

where the weight $w_j : \Omega_j \to \mathbb{R}$ is a continuous, nonnegative and compactly supported function with $\mathrm{supp}(w_j) \subseteq \Omega_j$. Then, we define the global RBF-PUM interpolant of the form

$$s(\boldsymbol{x}) = \sum_{j=1}^d s_j(\boldsymbol{x}) w_j(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega. \tag{3}$$

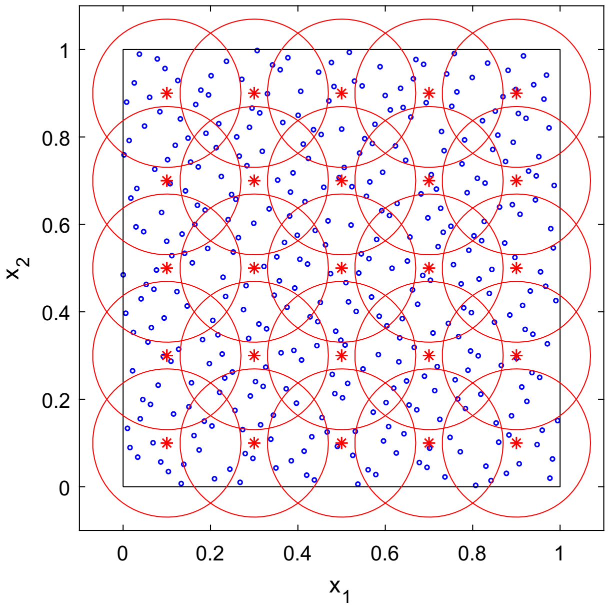

Fig. 1 Example of PU subdomains of fixed radius δ (red circles) with scattered data points (blue dots) in $\mathbb{R}^2$. Subdomain centers are denoted by red stars (Color figure online)

For each subdomain Ω_j, in (3) we can thus express the local RBF interpolants $s_j : \Omega_j \to \mathbb{R}$ as follows

$$s_j(\boldsymbol{x}) = \sum_{i=1}^{N_j} c_i^j \phi(||\boldsymbol{x} - \boldsymbol{x}_i^j||_2), \tag{4}$$

where N_j is the number of data points in Ω_j (i.e. the nodes $\boldsymbol{x}_i^j \in X_{N_j} = X_N \cap \Omega_j$), and we construct PU functions w_j using the well-known *Shepard's weight* [1,33]

$$w_j(\boldsymbol{x}) = \frac{\varphi_j(\boldsymbol{x})}{\sum_{k=1}^d \varphi_k(\boldsymbol{x})}, \quad j = 1, \ldots, d, \tag{5}$$

where $\varphi_j(\boldsymbol{x})$ is a compactly supported function with support on Ω_j such as the W2 function (see Table 1). Such functions are scaled with a shape parameter σ to get $\varphi_j(\boldsymbol{x}) = \varphi(\sigma||\boldsymbol{x} - \boldsymbol{\xi}_j||_2)$, $\boldsymbol{\xi}_j$ being the center of the weight function.

If the functions s_j, $j = 1, \ldots, d$, satisfy the interpolation conditions

$$s_j(\boldsymbol{x}_i^j) = f(\boldsymbol{x}_i^j), \quad \boldsymbol{x}_i^j \in \Omega_j, \quad i = 1, \ldots, N_j, \tag{6}$$

the global interpolant (3) inherits the interpolation property of the local interpolants (4), i.e.,

$$s(\boldsymbol{x}_i^j) = \sum_{j=1}^d s_j(\boldsymbol{x}_i^j) w_j(\boldsymbol{x}_i^j) = \sum_{j=1}^d f(\boldsymbol{x}_i^j) w_j(\boldsymbol{x}_i^j) = f(\boldsymbol{x}_i^j).$$

Solving the jth interpolation problem (6) results in the linear system

$$\begin{pmatrix} \phi(||\boldsymbol{x}_1^j - \boldsymbol{x}_1^j||_2) & \cdots & \phi(||\boldsymbol{x}_1^j - \boldsymbol{x}_{N_j}^j||_2) \\ \vdots & \vdots & \vdots \\ \phi(||\boldsymbol{x}_{N_j}^j - \boldsymbol{x}_1^j||_2) & \cdots & \phi(||\boldsymbol{x}_{N_j}^j - \boldsymbol{x}_{N_j}^j||_2) \end{pmatrix} \begin{pmatrix} c_1^j \\ \vdots \\ c_{N_j}^j \end{pmatrix} = \begin{pmatrix} f_1^j \\ \vdots \\ f_{N_j}^j \end{pmatrix},$$

or simply

$$\mathsf{A}_j \boldsymbol{c}_j = \boldsymbol{f}_j. \tag{7}$$

As for the global system (2), the use of SPD functions ϕ ensures (also in the local case) that the solution of the local system (7) exists uniquely, since the matrix A_j is nonsingular [17].

3.2 Selection of Local RBF Shape Parameters

Since the accuracy of the global interpolant (3) strongly depends upon the choice of shape parameter ε associated with the local RBFs in (4), we need an effective approach that enables us to find suitable values of ε (and, possibly, an optimal one for each of PU subdomains). The RBF shape parameter is in fact responsible for the flatness of the basis functions. However, in the *flat limit* $\varepsilon \to 0$, i.e. when the best accuracy is typically achieved, the local matrix in (7) might suffer from instability due to ill-conditioning (see [16]). As a consequence, the selection of ε may highly influence the accuracy of the RBF-PUM based interpolation. It is therefore paramount to optimally detect such values of the local RBF shape parameters.

3.2.1 Choice of ε via LOOCV

A good way to select a shape parameter ε is to use locally the LOOCV technique [30]. The idea behind LOOCV in the RBF-PUM interpolation is to split the data of each subdomain Ω_j, $j = 1, \ldots, d$, into two distinct sets:

– a *training set* $\{f(\boldsymbol{x}_1^j), \ldots, f(\boldsymbol{x}_{k-1}^j), f(\boldsymbol{x}_{k+1}^j), \ldots, f(\boldsymbol{x}_{N_j}^j)\}$,
– a *validation set* consisting of only the single value $f(\boldsymbol{x}_k^j)$ which was left out when creating the training set.

For a fixed index $k \in \{1, \ldots, N_j\}$ and a fixed shape parameter ε, we define the partial RBF interpolant

$$s_j^{[k]}(\boldsymbol{x}) = \sum_{i=1,\ i\neq k}^{N_j} c_i^j \phi(||\boldsymbol{x} - \boldsymbol{x}_i^j||_2),$$

whose coefficients c_i^j are found by interpolating the training data

$$s_j^{[k]}(\boldsymbol{x}_i^j) = f(\boldsymbol{x}_i^j), \qquad i = 1, \ldots, k-1, k+1, \ldots, N_j.$$

In order to measure the quality of this attempt, we define the error

$$e_k^j(\varepsilon) = f(\boldsymbol{x}_k^j) - s_j^{[k]}(\boldsymbol{x}_k^j) \tag{8}$$

at the one validation point $\boldsymbol{x}_k^j$ not used to determine the interpolant. The “optimal” value of ε is found as

$$\varepsilon_j^{opt} = \text{argmin}_\varepsilon ||\boldsymbol{e}_j(\varepsilon)||, \qquad \boldsymbol{e}_j = (e_1^j, \ldots, e_{N_j}^j)^T, \tag{9}$$

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

The important aspect is that we can determine the error vector $\boldsymbol{e}_j$ without solving N_j problems, each of size $(N_j-1)\times(N_j-1)$. In fact, instead of (8), the computation of the error components can be expressed in terms of the interpolation matrix A_j in (7), i.e.

$$e_k^j(\varepsilon) = \frac{c_k^j}{(\mathsf{A}_j^{-1})_{kk}}, \tag{10}$$

where c_k^j is the kth coefficient in the full RBF interpolant (4) and $(\mathsf{A}_j^{-1})_{kk}$ is the kth diagonal element of the matrix A_j^{-1} [18]. So from (9) and (10) it follows that the LOOCV cost function to be minimized is

$$\text{LOOCV}(\varepsilon) = ||\boldsymbol{e}_j(\varepsilon)||_\infty = \max_{k=1,\ldots,N_j} \left| \frac{c_k^j}{(\mathsf{A}_j^{-1})_{kk}} \right|. \tag{11}$$

3.2.2 Choice of $\boldsymbol{\varepsilon}$ via MLE

Another approach for selecting a shape parameter ε is given by the MLE, which relies on solid probabilistic and statistical foundations [18]; for further details, see e.g. [16,31,32]. As the LOOCV technique even a MLE based criterion can be applied to RBF-PUM interpolation to locally find an optimal ε-value for each subdomain Ω_j, $j=1,\ldots,d$.

In this stochastic context we introduce the concept of *random field* $Y_j = \{Y_j(\boldsymbol{x})\}_{\boldsymbol{x}\in\Omega_j}$ defined on the subdomain Ω_j. In particular, we assume that the field Y_j is a Gaussian random field, i.e., the vectors $\boldsymbol{Y}_j = (Y_j(\boldsymbol{x}_1), \ldots, Y_j(\boldsymbol{x}_{N_j}))^T$ of random variables have normal distributions with mean $\boldsymbol{\mu}_j = \mathbb{E}[\boldsymbol{Y}_j]$ and covariance matrix $\sigma_j^2\mathsf{A}_j = \text{Cov}(Y_j(\boldsymbol{x}_k, \boldsymbol{x}_i))_{k,i}^{N_j}$, where σ_j^2 is the process variance. Here, for simplicity we assume that $\boldsymbol{\mu}_j = \boldsymbol{0}$ and $\sigma_j^2 = 1$.

Now, for a fixed shape parameter ε, if the probability density that occurs an event given the data $\boldsymbol{f}_j$ is expressed as $p(\boldsymbol{f}_j|\varepsilon)$, then the function $\mathcal{L}(\varepsilon|\boldsymbol{f}_j)$ characterizes the *likelihood* of ε given the existing data values $\boldsymbol{f}_j$. Therefore, the Gaussian joint probability density function that refers to the vector $\boldsymbol{Y}_j$ of N_j random observations (belonging to the subdomain Ω_j) can be written as follows

$$p(\boldsymbol{Y}_j|\varepsilon) = \frac{1}{\sqrt{(2\pi)^{N_j}\det(\mathsf{A}_j)}} \exp\left(-\frac{1}{2}\boldsymbol{Y}_j^T\mathsf{A}_j^{-1}\boldsymbol{Y}_j\right). \tag{12}$$

When we evaluate the density function (12) by using the jth data vector $\boldsymbol{f}_j = (f^j(\boldsymbol{x}_1), \ldots, f^j(\boldsymbol{x}_{N_j}))^T$, we obtain the log-likelihood function[1]

$$\begin{aligned}\log\mathcal{L}(\varepsilon|\boldsymbol{f}_j) &= \log\left(\frac{\exp\left(-\frac{1}{2}\boldsymbol{f}_j^T\mathsf{A}_j^{-1}\boldsymbol{f}_j\right)}{\sqrt{(2\pi)^{N_j}\det(\mathsf{A}_j)}}\right)\\ &= -\frac{1}{2}\log(\det(\mathsf{A}_j)) - \frac{1}{2}\boldsymbol{f}_j^T\mathsf{A}_j^{-1}\boldsymbol{f}_j - \frac{N_j}{2}\log(2\pi).\end{aligned}$$

[1] We use the logarithm since the likelihood function is usually subjected to underflow and overflow.

In order to find the optimal ε parametrization, we thus need to maximize the log-likelihood function. Equivalently, we can multiply it by -2, and then—ignoring the constant term—minimize the resulting negative function.

If we instead want a criterion dependent of the variance process, we can obtain the following MLE cost function

$$\mathrm{MLE}(\varepsilon) = \log(\det(\mathsf{A}_j)) + N_j \log(\boldsymbol{f}_j^T \mathsf{A}_j^{-1} \boldsymbol{f}_j), \tag{13}$$

which can be minimized to determine the optimal value of ε (see [16]).

3.3 Convergence and Error Estimates

In order to formulate error bounds for RBF-PUM interpolation, we assume some additional regularity conditions on the subdomains Ω_j. In particular, we require that the PU weight functions w_j are *k-stable* [35]. This property holds if—besides assumptions given in Sect. 3.1—$\forall \boldsymbol{\alpha} \in \mathbb{N}_0^s$, with $|\boldsymbol{\alpha}| \leq k$, there exists a constant $C_{\boldsymbol{\alpha}} > 0$ such that

$$\left\| D^{\boldsymbol{\alpha}} w_j \right\|_{L_\infty(\Omega_j)} \leq \frac{C_{\boldsymbol{\alpha}}}{\delta_j^{|\boldsymbol{\alpha}|}}, \quad j = 1, \ldots, d,$$

where $\delta_j = \mathrm{diam}(\Omega_j) = \sup_{\boldsymbol{x},\boldsymbol{y} \in \Omega_j} ||\boldsymbol{x} - \boldsymbol{y}||_2$. Moreover, we need also to define the *fill distance*

$$h_{X_N,\Omega} = \sup_{\boldsymbol{x} \in \Omega} \min_{\boldsymbol{x}_i \in X_N} ||\boldsymbol{x} - \boldsymbol{x}_i||_2, \tag{14}$$

and make some further assumptions to have a *regular* covering $\{\Omega_j\}_{j=1}^d$ for (Ω, X_N) [36]. This condition demands that every subdomain Ω_j satisfies the so-called interior cone condition, and the local fill distances $h_{X_{N_j},\Omega_j}$ are uniformly bounded by the global fill distance (14).

After defining the space $C_\nu^k(\mathbb{R}^s)$ of all functions $f \in C^k$ whose derivatives of order $|\boldsymbol{\alpha}| = k$ satisfy $D^{\boldsymbol{\alpha}} f(\boldsymbol{x}) = \mathcal{O}(||\boldsymbol{x}||_2^\nu)$ for $||\boldsymbol{x}||_2 \to 0$, we consider the following convergence result, see [17, Theorem 29.1] and [36, Theorem 15.9].

Theorem 1 *Let $\Omega \subseteq \mathbb{R}^s$ be open and bounded and $X_N = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \subseteq \Omega$. Let $\phi \in C_\nu^k(\mathbb{R}^s)$ be a strictly conditionally positive definite function of order m. If $\{\Omega_j\}_{j=1}^d$ is a regular covering for (Ω, X_N) and $\{w_j\}_{j=1}^d$ is k-stable for $\{\Omega_j\}_{j=1}^d$, then the error between $f \in \mathcal{N}_\phi(\Omega)$ and its PUM interpolant (3) is bounded by*

$$|D^{\boldsymbol{\alpha}} f(\boldsymbol{x}) - D^{\boldsymbol{\alpha}} s(\boldsymbol{x})| \leq C h_{X_N,\Omega}^{(k+\nu)/2-|\boldsymbol{\alpha}|} |f|_{\mathcal{N}_\phi(\Omega)},$$

for all $\boldsymbol{x} \in \Omega$ and all $|\boldsymbol{\alpha}| \leq k/2$.

If we compare this convergence result with the global error estimates in [36], we see that the PUM preserves the local approximation order for the global fit (3). So we can efficiently compute large RBF interpolants by solving many small RBF interpolation problems and then glue them together with the global PU weights $\{w_j\}_{j=1}^d$ (see [17]).

4 Adaptive Bivariate Algorithm and Computational Issues

In this section we present the adaptive algorithm for bivariate interpolation of large scattered data sets. This adaptive approach is particularly useful when we are interested in solving interpolation problems that are characterized by unstructured or very irregularly distributed data. In this case, indeed, besides identifying suitable values of ε, in a PU framework it is important to have the chance of constructing subdomains of variable size. Since in this work we are considering radial kernels, it is natural to consider subdomains of circular shape. The search process is therefore directed to find a sufficient (or minimum) number of points to be able to define properly a local RBF interpolant. From a practical standpoint this means first to select in an adaptive way the radii δ of our subdomains, and then determine for each subdomain an optimal value of the shape parameter ε associated with the RBF. While the choice of ε has been discussed widely in Sect. 3.2, here we focus on the selection of suitable subdomain radii δ. This phase imposes to organize all given data in an efficient way so as to be able to quickly identify all the points belonging to the given subdomains. Indeed, when the initial radius of a subdomain is not large enough, i.e. the subdomain does not contain a sufficient number of points, the subdomain size has to increased thus including more points.

Now, in the following we give a description of our adaptive algorithm that enables us to find for each subdomain Ω_j, $j = 1, \ldots, d$, a couple of suitable values $(\varepsilon_j, \delta_j)$, also providing an analysis of the computational cost.

4.1 Data Structures and Search Procedures

In this subsection we describe how the data are organized in the two-dimensional space to select the nodes belonging to the various subdomains in the RBF-PUM based interpolation. By doing so, for the sake of clarity we usually assume that the search of points is carried out once (i.e., for a fixed index k), taking into account that an adaptive algorithm is obviously characterized by an iterative process. Computational efficiency is indeed an essential aspect to fast assembly the local matrix A_j in (7) and determine the corresponding local RBF interpolant (4). Hence here we present the search procedure used for the localization of points that lie in the subdomains Ω_j, $j = 1, \ldots, d$. Such a technique relies on a suitable partition of the domain Ω in square cells. Similar approaches have also been considered in [11,13]. Even if our partitioning structure is applicable to a generic domain, in this work we simply discuss the case of $\Omega = [0, 1]^2 \subseteq \mathbb{R}^2$.

First of all, we start with considering a cover of the domain Ω in which each subdomain Ω_j, $j = 1, \ldots, d$, has initially radius

$$\delta := \delta_j^{(0)} = \frac{1}{d_{PU}}, \tag{15}$$

where

$$d_{PU} = \left\lfloor \frac{1}{2}\sqrt{N} \right\rfloor \tag{16}$$

defines the number of PU centers along a single direction of Ω. The partition of the domain Ω is thus formed by $d = d_{PU}^2$ subdomains, whose centers are given by a grid of points (see the red stars in Fig. 1).

Remark 1 From the definition of d_{PU} in (16) it follows that if we take a larger (smaller) value of d, the partitioning structure becomes finer (coarser), i.e. less subdomains lead to larger ones (and vice versa). This specific connection between the parameters δ and d always ensures to be able to form a cover of Ω.

A choice as that given in (15) can be appropriate when we have a uniform or quite regular node distribution, but in case of irregularly distributed data the value (15) must be updated. In particular, we may iterate our adaptive process by computing the subdomain radii as follows

$$\delta_j^{(k)} = t_k \delta, \quad k = 1, 2, \dots, \tag{17}$$

where $t_k \in \mathbb{R}_{>1}$ is a value that increases with k until every subdomain Ω_j, $j = 1, \dots, d$, contains a number $N_j^{(k)}$ of points, which is larger than or equal to a prescribed minimum number $N_{\min}$[2].

Now, in order to localize all the nodes that lie in every subdomain Ω_j, we need to generate a data structure that enables us to partition suitably the domain Ω and the data points therein contained. Such a partition turns out to be particularly effective if it is combined with an efficient searching procedure. This search technique is based on a partition of Ω in b^2 square cells, where

$$b = \left\lceil \frac{1}{\delta} \right\rceil \tag{18}$$

denotes the cell number along one side of the domain. Thus we assume that the side of every square cell is equal to (or at most slightly less than) the initial subdomain radii. This choice enables us to examine in the searching procedure only a reduced number of cells, thus minimizing the computational effort w.r.t. the most advanced space-partitioning data structures, such as kd-trees, which are commonly used for range and nearest neighbor searches (see e.g. [2,6]). Acting in this way, given a generic point, say $\boldsymbol{x} = (x_1, x_2)$, we can compute the indexes

$$k_i = \left\lceil \frac{\boldsymbol{x}}{\delta} \right\rceil, \qquad i = 1, 2, \tag{19}$$

which identify the cell

$$k = (k_1 - 1)\, b + k_2, \tag{20}$$

in which the point $\boldsymbol{x}$ in (19) lies. Such a index is computed by taking into account the couple of cell coordinates (k_1,k_2) in (20) that move along x_1 and x_2 axes, respectively. It is therefore easy to assign to any point the corresponding cell k and partition a set of points in the domain Ω, as for instance the set X_N of data points. A sketch of the routine for building the cell based structure is shown in Procedure 1.

Procedure 1: Cell based Structure

```
STEP 1   For any point $\boldsymbol{x} \in X_N$
           (a) Compute the indexes $k_1$ and $k_2$ as in (19)
           (b) Find the cell $k$ in which $\boldsymbol{x}$ is located applying (20), and associate the index of $\boldsymbol{x}$ to it
STEP 2   Return the indexes belonging to each of the $b^2$ cells, i.e., all points of $X_N$ are identified in their
         own cell
```

Partitioning the domain Ω, we adopt a lexicographic order by proceeding from bottom to top and from left to right, numbering the square cells from 1 to b^2. Now, if a given subdomain Ω_j has radius (15) and its center lies in the kth cell, from (18) we deduce that the nodes belonging to Ω_j must be searched in the so-called *square neighborhood* consisted initially

[2] The number $N_j^{(k)}$ represents the amount of data points present in the subdomain Ω_j at the kth iteration; this value updates the generic definition of N_j introduced in (4), which has been used so far.

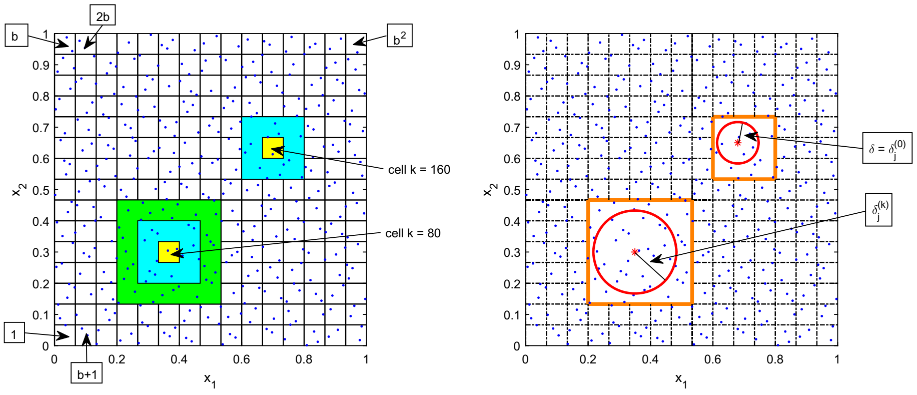

Fig. 2 Example of domain partition and search of points (blue dots) in the cell based structure. Left: cells $k = 80$ and $k = 160$ (yellow) together with one or two levels of neighboring cells (cyan and green). Right: square neighborhoods (orange) of various sizes including subdomains of radius $\delta = \delta_j^{(0)}$ (red, top-right) and $\delta_j^{(k)}$ (red, bottom-left), whose centers are denoted by the star * (Color figure online)

of nine cells, i.e. the kth square cell and its $3^2 - 1$ neighboring cells. However, when the number of points in Ω_j is not enough, we enlarge the radius as outlined in (17) and so the square neighborhood becomes larger as well. In so doing, in order to find all data points of Ω_j, we have to explore the kth cell and its $(3 + 2n)^2 - 1$ neighboring cells, for $n = 1, 2, \ldots$. In other words, for $n = 0$ we consider the first level (or "crown") of the neighborhood, for $n = 1$ the second one, and so on. An example of domain partition in square cells associated with the phase of localization and search of points within the adaptive subdomains is given in Fig. 2, left to right.

After partitioning the points in b^2 square cells, we have to answer the following inquiries, known respectively as *containing query* and *range search*. These computational issues can be briefly described as follows:

(i) given the center of a subdomain Ω_j, return the index k of the square cell in which that center is contained;
(ii) given a subdomain Ω_j and the nodes belonging to the corresponding square neighborhood, find all points that lie in that subdomain.

Therefore, the cell based containing query routine in item (i) provides the index k of the cell containing the subdomain center. Here, unlike (19), the cell coordinates k_1 and k_2 in (20) are

$$k_i = \left\lceil \frac{\boldsymbol{x}_j^c}{\delta} \right\rceil, \qquad i = 1, 2, \tag{21}$$

replacing the generic point $\boldsymbol{x}$ with the specific point $\boldsymbol{x}_j^c = (x_{j1}^c, x_{j2}^c)$, which represents the center of the subdomain Ω_j (see [13]). As regards item (ii) we have to construct beforehand the square neighborhoods, since we need to consider the points that are located in the given neighborhood only, instead of all points of the domain Ω. In the above description we observe that Ω_j is a generic subdomain, so we should think the subscript j is fixed. A practical example of the rule (20) for $k = 80$ (i.e., $k_1 = 6$ and $k_2 = 5$) and $k = 160$ (i.e., $k_1 = 11$ and $k_2 = 10$) is depicted in Fig. 2 (left). Then, after answering the first query, given a subdomain Ω_j the search routine enables us to determine all points that are contained in Ω_j. More precisely, since the center of such subdomain is located within the kth cell, the associated search technique identifies all data points which are in the kth cell and in its neighboring cells, see Fig. 2 (right). The algorithm also provides for the chance to reduce the number of neighboring cells to be examined when a cell is placed close to the boundary of the domain Ω. An outline of the routines for solving containing query and range search problems is illustrated in Procedure 2 and Procedure 3, respectively, while a summary of the whole adaptive algorithm for bivariate RBF-PUM interpolation is sketched in Algorithm 1.

Procedure 2: Containing Query

```
STEP 1    Define the subdomain center of $\Omega_j$
STEP 2    Compute the cell coordinates $(k_1, k_2)$ given in (21)
STEP 3    Apply the rule (20) and obtain the index $k$ of the cell containing the center of $\Omega_j$
```

Procedure 3: Range Search

```
STEP 1    Define the square neighborhood associated with a subdomain$\Omega_j$
STEP 2    Compute the Euclidean distance between the center of $\Omega_j$ and all data points belonging to the square neighborhood
STEP 3    Sort all computed distances and return all points contained in the subdomain $\Omega_j$
```

4.2 Complexity Analysis

In this subsection we discuss computational complexity of the adaptive algorithm. To keep the presentation easier, in the following we explicitly refer to the various steps of Algorithm 1. First of all, we observe that STEPS 1, 2 and 3 are essentially preliminary phases, where we define the basic setup for the RBF-PUM and the related partition in cells of the domain Ω. This stage does not significantly influence the computational cost. Then, in STEP 4 we build and apply the cell based structure to assign the corresponding cell to each interpolation point of X_N. This step has a $\mathcal{O}(N)$ cost. In the assessment of the total complexity we should however take into account the cost which derives from the storing of other points (e.g., evaluation points) or the number d of subdomains used to construct the PUM. In STEP 5 for every subdomain Ω_j, $j = 1, \ldots, d$, we have to solve containing query and range search problems; in this phase, the process also needs to compute a square neighborhood consisting of nine cells, i.e. the ones at zero and one levels, respectively (see yellow and cyan cells in Fig. 2, left). Now, applying the containing query and range search routines, the complexity is of the order $\mathcal{O}(N)$. The procedure is then iterated in STEP 6 until the while-loop has reached the minimum number of prescribed points in each subdomain Ω_j. In an adaptive process this could cause an increase of the computational cost, in particular when we have unstructured or significantly different data distributions within the domain. In fact, this stage establishes to update the subdomain size and, accordingly, to enlarge the square neighborhood. Here, we can also observe that the call to the range search routine requires a sort of all computed distances

Algorithm 1: Adaptive RBF-PUM Algorithm

```
STEP 1   Create a cover of the domain $\Omega$ with subdomains $\Omega_j$ of radius (15)
STEP 2   Define a partition of $\Omega$ in $b^2$ square cells, where $b$ is given in (18)
STEP 3   Fix the minimum number $N_{\min}$ of points required in each subdomain $\Omega_j$
STEP 4   Apply the cell based structure for the identification of points in the $b^2$ cells
STEP 5   For each $\Omega_j$, find the cell containing the subdomain center (containing query)
             and, generated the square neighborhoods formed by $3^2$ cells,
             determine all points that lie in $\Omega_j$ (range search)
    STEP 6   While $N_j^{(k)} < N_{\min}$, with iteration $k = 1, 2, \dots$
               a) Compute the subdomain radii (17)
               b) Enlarge the square neighborhoods consisted of $(3+2n^*)^2$ cells,
                    $n^* \in \{1, 2, \dots\}$, and find additional points of $\Omega_j$ (range search)
    STEP 7   Select the local RBF shape parameter via LOOCV or MLE
    STEP 8   Compute local interpolants (4) and weight functions (5)
STEP 9   Evaluate the global interpolant (3)
```

within the subdomains Ω_j, $j = 1, \dots, d$ (see STEP 3 of Procedure 3). However, due to local use of a quicksort routine whose complexity is $\mathcal{O}(N_j \log N_j)$, the cost of this phase is estimated to be $\mathcal{O}(1)$. In STEP 7, the algorithm finds automatically the shape parameters associated with the local RBF interpolants. As discussed in Sect. 3.2, the choice of ε can be done, either using LOOCV or MLE. Both methods require computation of the inverse matrix A_j^{-1}, $j = 1, \dots, d$ with a cost of $\mathcal{O}(N_j^3)$. Even so, the actual selection of the ε-parameter is carried out in an efficient way by minimizing the cost functions (11) or (13) via the MATLAB `fminbnd` optimization routine. Finally, in STEPS 7 and 8 we have to add up a constant number of local interpolants and weight functions to evaluate the global fit (3). This last stage can be completed with a complexity of $\mathcal{O}(1)$.

5 Numerical Results

In this section we illustrate the performance of our adaptive algorithm, which is implemented in MATLAB environment. All the numerical experiments have been carried out on a laptop with an Intel(R) Core i7 6500U CPU 2.50GHz processor and 8.00GB RAM, while the results are shown in tables and figures.

In the following we focus on a wide series of experiments carried out by running the adaptive algorithm for 2D RBF-PUM interpolation. This study aims to analyze the algorithm behavior when the use of an adaptive scheme is essential to get reliable results in broad sense. In doing so, in our tests we consider four different sets of irregularly distributed (or scattered) data points contained in the unit square $\Omega = [0, 1]^2 \subset \mathbb{R}^2$. Each of these unstructured node distributions exhibits distinct features. The latter have been chosen to stress the adaptive method and see how the numerical algorithm works in various situations. For the sake of simplicity, we denote these four data sets with the names “Halton”, “Ameoba”, “Starfish” and “Strips”, which are defined as follows:

– “Halton” refers to a data set consisting of $N = 4\,096$ low discrepancy Halton points [37] generated through the MATLAB command `haltonset(2,'Skip',1)`, see Fig. 3 top-left;
– “Ameoba” is a data set characterized by five different distributions, with $N = 8\,419$. In the middle of the domain we have a node distribution with 4 460 nodes within an Ameoba like shape region which is bounded by the parametric curve [34]

$$r(\theta) = e^{\sin(\theta)} \sin^2(2\theta) + e^{\cos(\theta)} \cos^2(2\theta), \qquad \theta \in [0, 2\pi),$$

while the remaining area is split into four small “incomplete” squares containing 165, 257, 438 and 3 099 points, respectively, see Fig. 3 top-right;
– “Starfish” identifies a data set containing four distributions of points with distinct densities in the domain and on the whole $N = 11\,436$ interpolation nodes. In the central part of the domain we have a starfish like shape area with 2 547 points bounded by the parametric curve [24]

$$r(\theta) = 0.8 + 0.1(\sin(6\theta) + \sin(3\theta)), \qquad \theta \in [0, 2\pi).$$

The other parts of Ω are characterized by three oblique bands in which, excluding the starfish-like region, there are 1 287, 5 461 and 2 141 points, respectively, see Fig. 3 bottom-left;
– “Strips” labels the last data set with $N = 14\,001$ points. In this case we consider five vertical strips, each of them having a different node distribution. More precisely, left-to-right, we move going from a low density to a high density of points, which is proved by the fact that strips of equal area contain 802, 1 800, 2 801, 3 800, and 4 798 points, respectively, see Fig. 3 bottom-right.

Besides selecting suitable subdomains of variable size in the PUM as discussed in Sect. 4, the adaptive algorithm also ensures dependable previsions of the RBF shape parameters via LOOCV or MLE (see Sect. 3.2). The detection of such parameters is completely automatic and the ε-computation is done by using the MATLAB `fminbnd` minimization routine. Moreover, in (17) we assume that the value $t_k = 1 + k/8$, with $k = 1, 2, \ldots$, demanding that each subdomain contains at least $N_{\min} = 15$ data points. We thus show the results obtained by applying our adaptive PUM algorithm and using as local interpolants in (4) some of the SPD RBFs contained in Table 1. As a matter of fact, since we are interested in studying in depth how much our method is effective, the analysis is based on considering various local kernels, thus involving radial functions of different smoothness such as GA, IMQ, W6, M6, M4 and M2. In regard to Shepard’s weight in (5) we take the compactly supported function W2.

In these experiments we analyze the performance of our algorithm taking the data values by three test functions. The former is known as Franke’s function [26], and its analytic expression is

$$f_1(x_1, x_2) = \frac{3}{4}\mathrm{e}^{-\frac{(9x_1-2)^2+(9x_2-2)^2}{4}} + \frac{3}{4}\mathrm{e}^{-\frac{(9x_1+1)^2}{49} - \frac{9x_2+1}{10}} + \frac{1}{2}\mathrm{e}^{-\frac{(9x_1-7)^2+(9x_2-3)^2}{4}} - \frac{1}{5}\mathrm{e}^{-(9x_1-4)^2-(9x_2-7)^2}.$$

The latter is a trigonometric function [29] of the form

$$f_2(x_1, x_2) = 2\cos(10x_1)\sin(10x_2) + \sin(10x_1x_2),$$

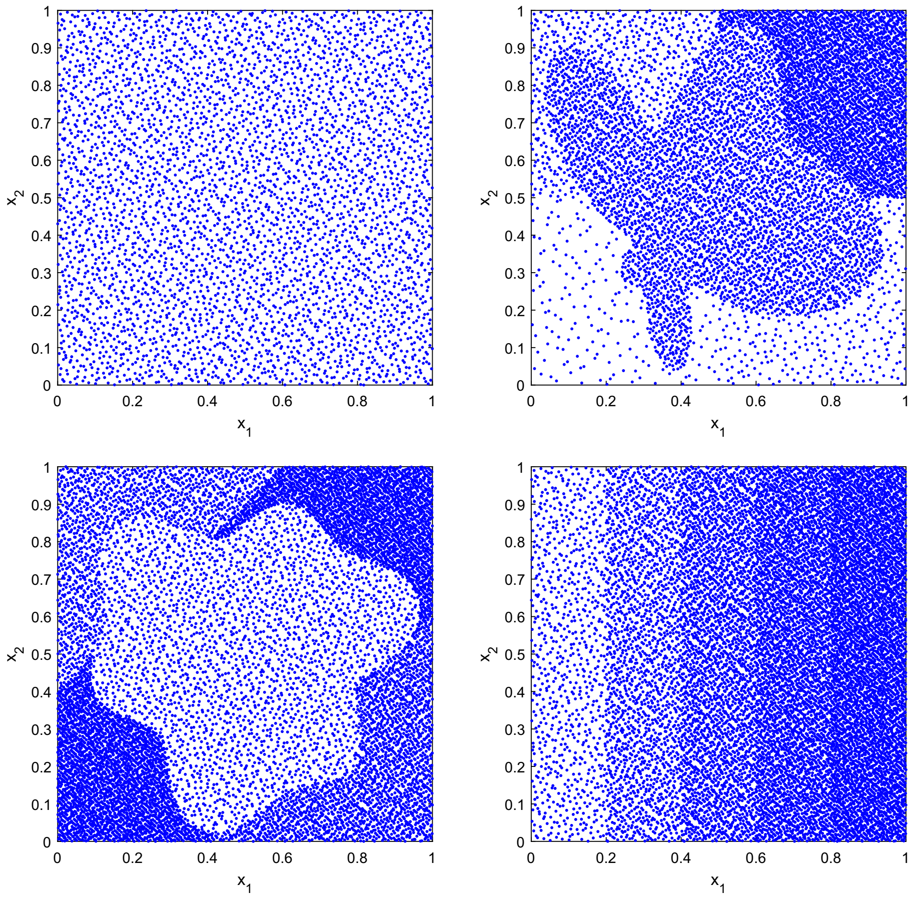

Fig. 3 Graphical representation of data sets used for 2D interpolation: “Halton” (top-left), “Ameoba” (top-right), “Starfish” (bottom-left) and “Strips” (bottom-right)

while the last one [7,21] is given by

$$f_3(x_1, x_2) = \frac{1}{2} x_2 \cos^4 \left[4 \left(x_1^2 + x_2 - 1 \right) \right].$$

In Fig. 4 we show a graphical representation of the above functions, which are commonly used to test and validate new methods and algorithms, then making them usable in several fields of applied sciences and engineering.

In order to investigate accuracy of the interpolation method, we compute the Root Mean Square Error (RMSE), whose formula is given by

$$\text{RMSE} = \sqrt{\frac{1}{N_{\text{eval}}} \sum_{i=1}^{N_{\text{eval}}} |f(\boldsymbol{\xi}_i) - s(\boldsymbol{\xi}_i)|^2} = \frac{1}{\sqrt{N_{\text{eval}}}} ||f - s||_2, \tag{22}$$

where the $\boldsymbol{\xi}_i$, $i = 1, \ldots, N_{\text{eval}}$, are a grid of evaluation points. Here, we assume $N_{\text{eval}} = 40 \times 40$.

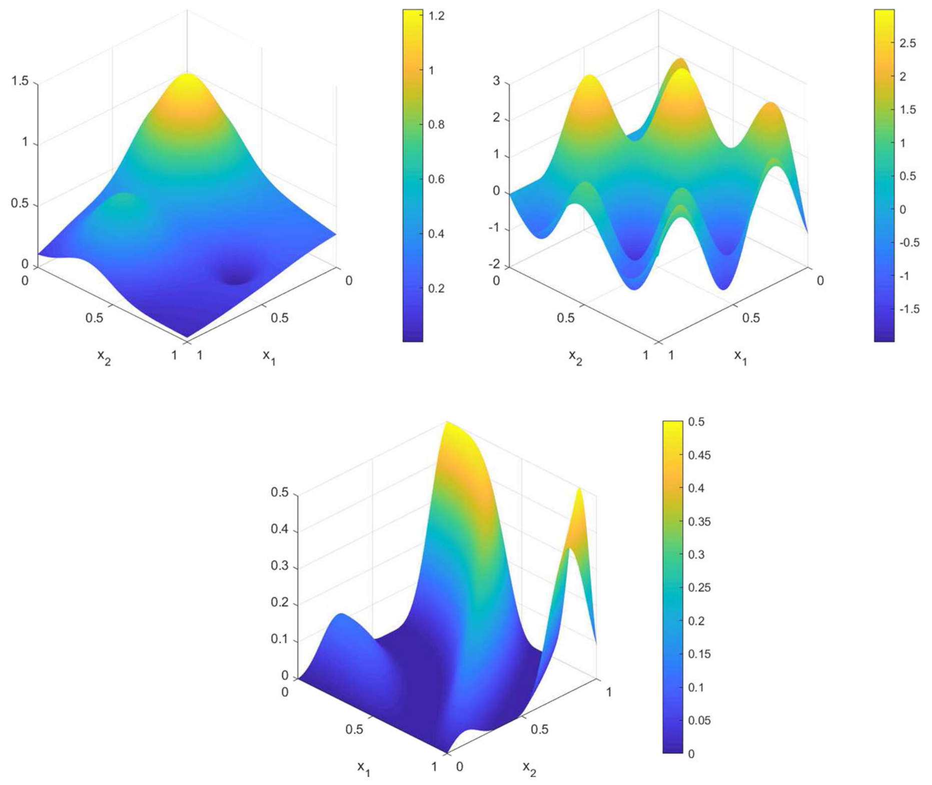

Fig. 4 Graphical representation of test functions f_1 (top, left), f_2 (top, right) and f_3 (bottom, center)

The target of our study is therefore two-fold: on the one hand, studying what is the level of accuracy that this adaptive interpolation method can achieve; on the other, analyzing the computational efficiency expressed in terms of CPU times (in seconds) of the proposed algorithm. In order to emphasize the benefit deriving from this new numerical code, we compare our implementation with a non-adaptive algorithm and an adaptive one characterized by the use of standard search procedures, i.e. without using the procedures discussed in Sect. 4.1.

First of all, we start with an analysis on the accuracy of our adaptive RBF-PUM interpolation scheme. Thus, in Tables 2, 3, and 4, we report the computation errors obtained by applying our algorithm on the four data sets and the three test functions previously mentioned. This study enables us to see how good the ε-predictions via LOOCV and MLE within our local method are. From this comparison it is quite clear that LOOCV seems to be able to provide a greater effectiveness, since its use usually leads to more accurate results than MLE. This fact is especially evident when a high regularity kernel such as GA is employed, instead of a limited smoothness kernel like M4. However, we can also observe that in general the LOOCV gives a slight improvement in term of precision, even if in some cases the benefit is around a order of magnitude. For this reason, in our next tests we will mainly focus on use of LOOCV technique.

Then, in Tables 5 and 6, our focus is to show the importance of our adaptive algorithm with respect to a non-adaptive one. This relevance is undeniable in terms of accuracy of the

Table 2 RMSEs obtained by applying the adaptive algorithm for f_1

Data set	GA		M4	
	LOOCV	MLE	LOOCV	MLE
Halton	1.22e−6	3.57e−5	1.19e−5	6.47e−5
Ameoba	1.48e−5	2.93e−4	1.86e−4	4.27e−4
Starfish	6.35e−7	9.31e−6	3.72e−6	1.19e−5
Strips	4.27e−7	2.11e−5	1.26e−5	6.04e−5

Table 3 RMSEs obtained by applying the adaptive algorithm for f_2

Data set	GA		M4	
	LOOCV	MLE	LOOCV	MLE
Halton	1.68e−5	1.78e−4	4.03e−4	3.81e−4
Ameoba	1.47e−4	1.65e−3	1.93e−3	2.45e−3
Starfish	4.46e−6	4.23e−5	6.68e−5	9.36e−5
Strips	1.12e−5	1.37e−4	2.07e−4	2.02e−4

Table 4 RMSEs obtained by applying the adaptive algorithm for f_3

Data set	GA		M4	
	LOOCV	MLE	LOOCV	MLE
Halton	1.97e−5	3.04e−5	1.33e−4	1.36e−4
Ameoba	1.38e−5	6.26e−5	3.64e−5	4.48e−5
Starfish	2.07e−6	5.27e−6	2.10e−5	2.47e−5
Strips	8.20e−7	5.75e−6	1.80e−5	2.49e−5

numerical method (in these tests the PUM uses IMQ as local kernel). In fact, the benefit deriving from adaptivity is noteworthy, not only when very irregularly distributed data sets (e.g., Ameoba, Starfish and Strips) are considered but also in case of quasi-random data points (Halton), see Fig. 3. In particular, from previous tables we highlight that the adaptive algorithm gives results that—when they are computable—are about two or three orders of magnitude more accurate than the non-adaptive one. Indeed, we note as the non-adaptive interpolation algorithm can be applied successfully only in two cases (Halton and Starfish), while in the other two situations (Ameoba and Strips) is not possible to get any result. This drawback is essentially due to the fact that the non-adaptive method cannot find points in every subdomain, thus making the interpolation process not practicable. In Tables 5 and 6 we denote this computational issue with the symbol –. As regards the computational efficiency expressed in CPU times, the adaptive implementation is obviously a little more costly than the non-adaptive one. Nevertheless, as outlined from our experiments, this extra-work is quite limited and fully compensated by high level of reliability of the new adaptive algorithm.

In Fig. 5 we study the behavior of interpolation errors (RMSEs) and execution times (CPU times) by varying the minimum number $N_{\min}$ of data points that is required to lie in each subdomain. In this analysis, for shortness, we focus our attention on a specific case, comparing the behavior of LOOCV and MLE techniques for fixed values of $N_{\min} \in \{10, 12, \ldots, 30\}$. Here the algorithm is tested on the "Strips" data set for f_1, using M6 as a local kernel. Figure 5 (left) confirms once more that LOOCV results in greater accuracy than MLE; at the same time, these tests show that the highest level of precision due to LOOCV is obtained

Table 5 Interpolation errors and execution times (in seconds) obtained by comparing our new adaptive algorithm with a non-adaptive implementation for f_1. Both RBF-PUM algorithms are tested by using IMQ and selecting ε-values via LOOCV

Data set	Adaptive algorithm		Non-adaptive algorithm	
	RMSE	CPU time	RMSE	CPU time
Halton	1.75e−6	3.45	9.33e−5	2.50
Ameoba	1.99e−5	7.09	–	–
Starfish	7.06e−7	9.03	4.14e−4	6.96
Strips	4.64e−7	10.44	–	–

Table 6 RMSEs obtained by applying our new adaptive algorithm and a non-adaptive implementation. Both RBF-PUM interpolation algorithms are tested by using IMQ and selecting ε-values via LOOCV

Data set	f_2		f_3	
	adaptive	non-adaptive	adaptive	non-adaptive
Halton	2.45e−5	1.74e−3	2.07e−5	4.87e−4
Ameoba	2.32e−4	–	1.25e−5	–
Starfish	7.80e−6	6.55e−3	2.19e−6	4.88e−4
Strips	1.90e−5	–	1.53e−6	–

for values of $N_{\min}$ between 14 and 26, while we observe a quite uniform error behavior for MLE. From Fig. 5 (right), as expected, we can also note that for both LOOCV and MLE the CPU times grow as $N_{\min}$ increases, thus making the algorithm computationally more expensive. The given results are not immutable facts, because they are obviously influenced by several variables present in these numerical experiments (e.g., local kernel, test function, data set, etc.). However, this analysis offers useful information for the choice of appropriate values of $N_{\min}$. Moreover, it points out that the selected value $N_{\min} = 15$ turns out to be a good choice for our purposes. This statement is also true if we analyze results contained in Fig. 6, where we use the LOOCV based estimator and compare M6 and W6 to see how RMSEs and CPU times change by varying $N_{\min}$. In this case, we report graphs obtained by running our algorithm on the "Halton" data set for f_3. From these results we notice that the accuracy of globally supported M6 is slightly better than compactly supported W6. The same considerations can be extended—even in a more pronounced way—for the execution time.

Finally, in Tables 7 and 8 we further test our adaptive interpolation scheme on the test function f_1. More precisely, here we are interested in comparing execution times obtained by running our new adaptive algorithm and a standard one characterized by the use of the MATLAB `rangesearch` routine, instead of the cell based procedures discussed in Sect. 4. Both algorithms make use of LOOCV for the ε-selection. In Table 7 we report the results computed with the M2 kernel and for the four data sets introduced at the beginning of this section, while in Table 8 we consider the local kernel M4 and five sets of Halton data points whose number N goes from 1 089 to 263 169. Especially looking at Table 8, we may highlight the great speed-up between the two algorithms, underlining as this gap tends to be more and more remarkable when the number of interpolation nodes increases. These results show the significant improvement in terms of computational efficiency that the use of our adaptive algorithm produces.

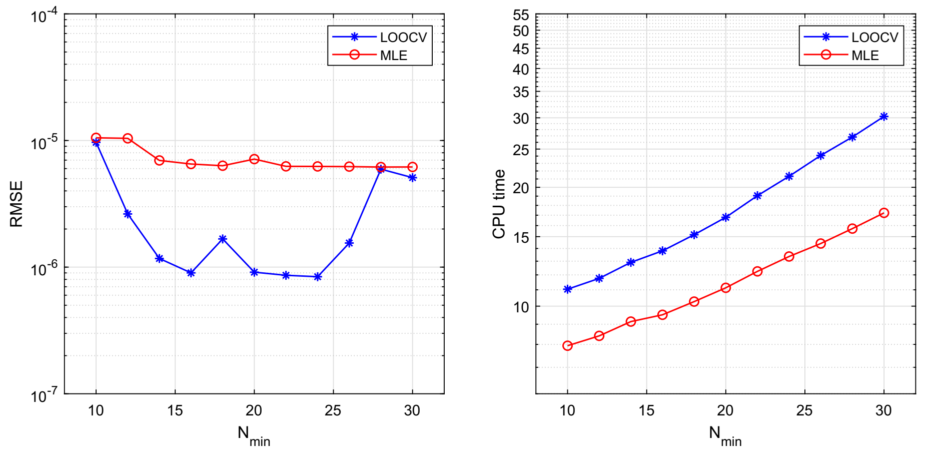

Fig. 5 RMSEs (left) and CPU times (right) obtained by varying the value of $N_{\min}$; in this comparison between LOOCV and MLE, our adaptive algorithm uses M6 as a local kernel and is applied on the "Strips" data set for f_1

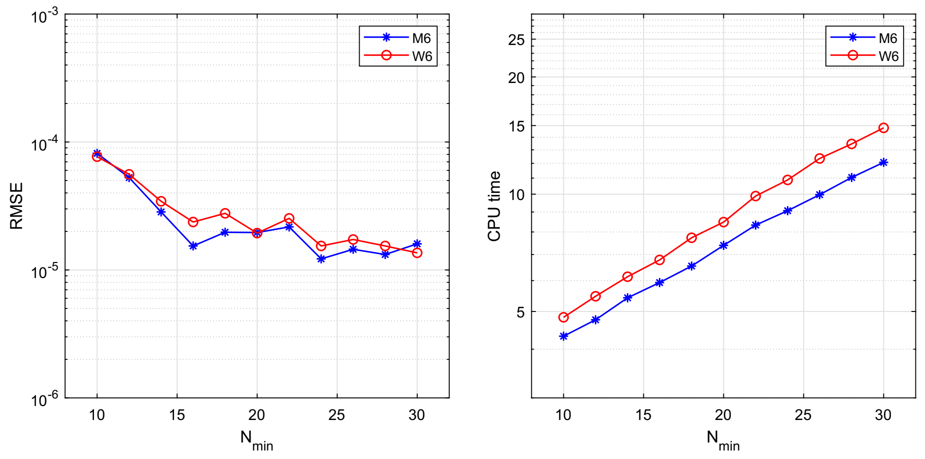

Fig. 6 RMSEs (left) and CPU times (right) obtained by varying the value of $N_{\min}$; in this comparison between M6 and W6, our adaptive algorithm uses LOOCV as an ε-estimator and is applied on the "Halton" data set for f_3

Table 7 RMSEs and CPU times (in seconds) computed by applying LOOCV and using M2 for f_1

Data set	RMSE	CPU time		Speed-up
		New algorithm	Standard algorithm	
Halton	1.12e−4	5.03	22.20	4.41
Ameoba	5.02e−4	10.65	149.29	14.02
Starfish	3.01e−5	13.78	162.03	11.76
Strips	9.71e−5	16.15	316.82	19.57

Execution times are obtained by comparing our new adaptive algorithm with a standard implementation characterized by the use of the MATLAB `rangesearch` routine, instead of that discussed in Sect. 4; in the latter case, the speed-up (ratio of execution times) between the standard algorithm and the new one is given

Table 8 RMSEs and CPU times (in seconds) computed on Halton points by applying LOOCV and using M4 for f_1

N	RMSE	CPU time		Speed-up
		New algorithm	Standard algorithm	
1 089	1.04e−4	1.54	3.86	2.51
4 225	1.17e−5	5.11	21.00	4.11
16 641	1.33e−6	17.33	232.60	13.18
66 029	1.67e−7	63.24	3 635.12	57.48
263 169	1.90e−8	223.38	58 041.40	259.83

Execution times are obtained by comparing our new adaptive algorithm with a standard implementation characterized by the use of the MATLAB `rangesearch` routine, instead of that discussed in Sect. 4; in the latter case, the speed-up (ratio of execution times) between the standard algorithm and the new one is given

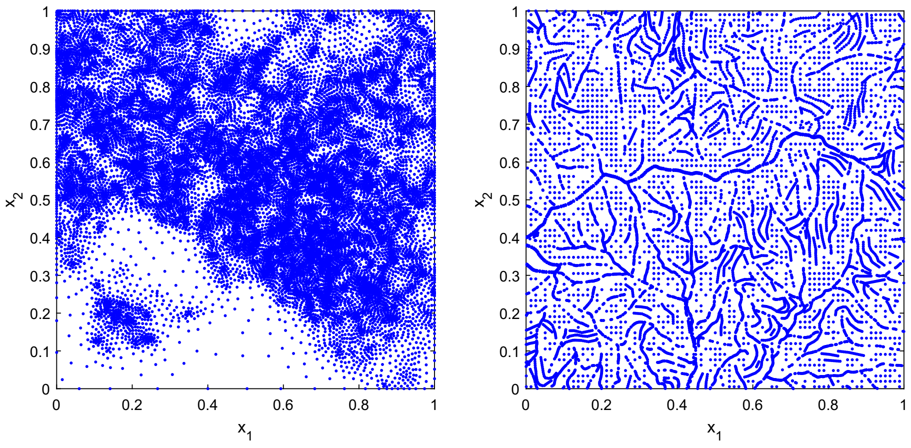

Fig. 7 Graphical representation of Black Forest (left) and Gattinara (right) data sets

6 Applications

In this section we test our adaptive algorithm on two real world data sets. In the first example, we consider an application oriented to approximate the *Black Forest* elevation data set [8,14], which consists of 15 885 data points. This data set refers to a region in the neighborhood of Freiburg (Germany). It represents a specific case of scattered data with highly varying density as shown in Fig. 7 (left). In the second example, we focus on the approximation of the so-called *Gattinara* topography data set, which is characterized by 10 671 data points. The latter belong to the homonymous geographic area, close to the city of Gattinara in province of Vercelli (Italy). Also in this situation, although the data set distribution is quite different from the Black Forest one, we have a typical case of very irregularly distributed data points as evident from Fig. 7 (right). Both regions are mountain areas: the differences in height are 1214 m and 309.87 m, respectively, while minimum and maximum heights for such data are gathered in Table 9. In Fig. 8 we report a 3D view for the Black Forest (left) and Gattinara (right) data sets.

Since we are working with real data (and therefore we do not have any exact or true solution), we assess reliability of our results by considering a technique that is commonly

Table 9 Minimum and maximum heights (in meters) in the real world data sets

Data set	Height min	max
Black Forest	172.875	1 386.9
Gattinara	247.38	557.25

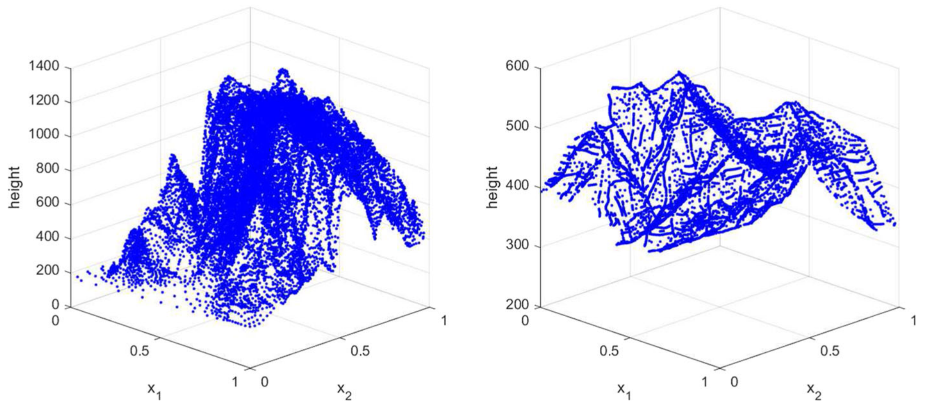

Fig. 8 3D view of Black Forest (left) and Gattinara (right) data sets

Table 10 RMSEs (in meters) and CPU times (in seconds) obtained by applying our new adaptive algorithm on real world data sets. Tests have been done by using M2 and $N_{\min} = 25$

Data set	LOOCV RMSE	CPU time	MLE RMSE	CPU time
Black Forest	5.56	22.58	5.56	11.45
Gattinara	2.30	13.60	2.35	5.90

used in applications. For Black Forest data set we have 15 885 elevation data points, which we split into two subsets: first, we randomly select $N = 15\,715$ nodes for the RBF-PUM interpolation process; second, we reserve the remaining $N_{\text{eval}} = 170$ evaluation points for the cross validation. Roughly in the same way, we act for Gattinara data set. We start from the 10 671 points, and then we subdivide this data set by taking $N = 10\,600$ interpolation nodes and $N_{\text{eval}} = 71$ evaluation points. In Table 10, we report the numerical results obtained by applying our adaptive algorithm. The latter are computed by using M2 as local RBF interpolant and selecting the local shape parameters via LOOCV or MLE, with $N_{\min} = 25$. From this table we can observe as LOOCV and MLE provide a very similar accuracy, with a slight prevalence of LOOCV, while—as already outlined in numerical experiments of Sect. 5—the MLE is about twice faster than LOOCV. Note that, although such errors are larger than the ones shown in Sect. 5, they turn out to be consistent with the previous results; in fact, in these real world situations, the error in (22) is measured in meters.

From more extensive tests we can also point out that in the Black Forest case LOOCV and MLE seem to have a similar predictive capability because the same level of accuracy is achieved, even when $N_{\min}$ varies between 10 and 30 (see Fig. 9, left). For the Gattinara data set we can observe that the MLE is more accurate than LOOCV for smaller values of $N_{\min}$, whereas this behavior is reversed for larger ones (see Fig. 9, right). Instead, as regards

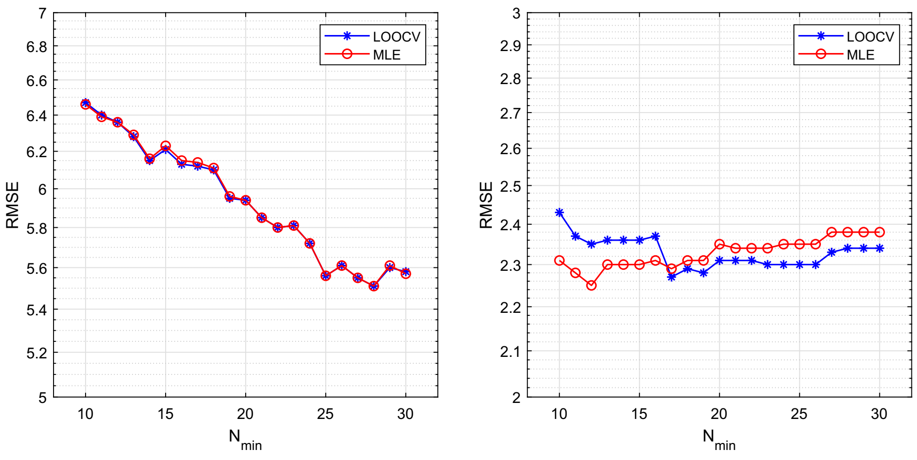

Fig. 9 RMSEs obtained by using M2 and varying the value of N_{min} for Black Forest (left) and Gattinara (right) data sets

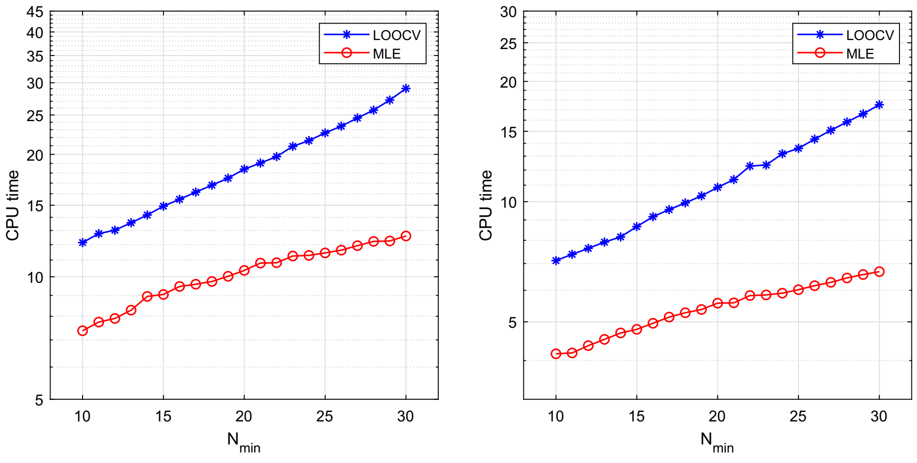

Fig. 10 CPU times obtained by using M2 and varying the value of N_{min} for Black Forest (left) and Gattinara (right) data sets

execution times (with equal parameters) MLE is constantly more efficient than LOOCV from the computational standpoint (see Fig. 10).

7 Conclusions and Future Work

In this paper we proposed a new adaptive algorithm for bivariate interpolation of large scattered data points through the RBF-PUM. We showed performance and efficacy of our numerical method by solving interpolation problems with artificial and real data sets, which were very irregularly distributed or with highly varying density in the domain. Compared to non-adaptive or standard RBF-PUM implementations, this adaptive algorithm enabled us to achieve accurate solutions for problems which in some cases might be unsolvable, also significantly reducing computational cost and execution time. All these results have been obtained by mainly exploiting the meshfree nature of the RBF-PUM. We thus created an adaptive scheme with subdomains of variable size, implementing an efficient search procedure for the localization of data points. The adaptive algorithm has been devised to effectively find optimal values of the RBF shape parameters by using LOOCV or MLE based criteria. This choice makes the scheme entirely automatic.

As future work we propose to further enhance our adaptive algorithm, for example optimizing the selection of the minimal number of points within each subdomain. Further studies in this direction will be discussed in future works.

Acknowledgements The author sincerely thanks the editor and the anonymous referee for insightful comments and suggestions, which gave the chance to significantly improve the quality of this paper. This work was partially supported by the INdAM-GNCS 2020 research project "Multivariate approximation and functional equations for numerical modeling" and by the 2020 project "Mathematical methods in computational sciences" funded by the Department of Mathematics "Giuseppe Peano" of the University of Torino. This research has been accomplished within the RITA "Research ITalian network on Approximation" and the UMI Group TAA "Approximation Theory and Applications". The author is member of the INdAM Research group GNCS.

Funding Open access funding provided by Università degli Studi di Torino within the CRUI-CARE Agreement.

References

1. Allasia, G., Cavoretto, R., De Rossi, A.: Hermite-Birkhoff interpolation on scattered data on the sphere and other manifolds. Appl. Math. Comput. **318**, 35–50 (2018)
2. Arya, S., Mount, D., Netanyahu, N., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J. ACM **45**, 891–923 (1998)
3. Babuška, I., Melenk, J.M.: The partition of unity method. Int. J. Numer. Methods Eng. **40**, 727–758 (1997)
4. Ben-Ahmed, E.H., Sadik, M., Wakrim, M.: Radial basis function partition of unity method for modelling water flow in porous media. Comput. Math. Appl. **75**, 2925–2941 (2018)
5. Ben-Ahmed, E.H., Sadik, M., Wakrim, M.: A stable radial basis function partition of unity method with d-rectangular patches for modelling water flow in porous media. J. Sci. Comput. **84**, 18 (2020)
6. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry. Springer, Berlin (1997)
7. Bozzini, M., Lenarduzzi, L., Rossini, M.: Polyharmonic splines: An approximation method for noisy scattered data of extra-large size. Appl. Math. Comput. **216**, 317–331 (2010)
8. Bracco, C., Giannelli, C., Sestini, A.: Adaptive scattered data fitting by extension of local approximations to hierarchical splines. Comput. Aided Geom. Design **52–53**, 90–105 (2017)
9. Buhmann, M.D.: Radial Basis Functions: Theory and Implementation, Cambridge Monographs on Applied and Computational Mathematics, vol. 12. Cambridge University Press, Cambridge (2003)
10. Cavoretto, R.: A numerical algorithm for multidimensional modeling of scattered data points. Comput. Appl. Math. **34**, 65–80 (2015)
11. Cavoretto, R., De Rossi, A.: A trivariate interpolation algorithm using a cube-partition searching procedure. SIAM J. Sci. Comput. **37**, A1891–A1908 (2015)
12. Cavoretto, R., De Rossi, A.: Adaptive meshless refinement schemes for RBF-PUM collocation. Appl. Math. Lett. **90**, 131–138 (2019)

13. Cavoretto, R., De Rossi, A., Perracchione, E.: Optimal selection of local approximants in RBF-PU interpolation. J. Sci. Comput. **74**, 1–22 (2018)
14. Davydov, O., Zeilfelder, F.: Scattered data fitting by direct extension of local polynomials to bivariate splines. Adv. Comput. Math. **21**, 223–271 (2004)
15. Driscoll, T., Heryudono, A.: Adaptive residual subsampling methods for radial basis function interpolation and collocation problems. Comput. Math. Appl. **53**, 927–939 (2007)
16. Fasshauer, G., McCourt, M.: Kernel-based Approximation Methods using Matlab, Interdisciplinary Mathematical Sciences, vol. 19. World Scientific, Singapore (2015)
17. Fasshauer, G.E.: Meshfree Approximation Methods with Matlab, Interdisciplinary Mathematical Sciences, vol. 6. World Scientific, Singapore (2007)
18. Fasshauer, G.E.: Positive definite kernels: Past, present and future. Dolomites Res. Notes Approx. **4**, 21–63 (2011)
19. Fereshtian, A., Mollapourasl, R., Avram, F.: RBF approximation by partition of unity for valuation of options under exponential Lévy processes. J. Comput. Sci. **32**, 44–55 (2019)
20. Fornberg, B., Flyer, N.: A Primer on Radial Basis Functions with Applications to the Geosciences. SIAM, Philadelphia (2015)
21. Franke, R., Hagen, H.: Least squares surface approximation using multiquadrics and parametric domain distorsion. Comput. Aided Geom. Design **16**, 177–196 (1999)
22. Gholampour, F., Hesameddini, E., Taleei, A.: A stable RBF partition of unity local method for elliptic interface problems in two dimensions. Eng. Anal. Bound. Elem. **123**, 220–232 (2021)
23. Heryudono, A., Larsson, E., Ramage, A., von Sydow, L.: Preconditioning for radial basis function partition of unity methods. J. Sci. Comput. **67**, 1089–1109 (2016)
24. Larsson, E., Lehto, E., Heryudono, A., Fornberg, B.: Stable computation of differentiation matrices and scattered node stencils based on gaussian radial basis functions. SIAM J. Sci. Comput. **35**, A2096–A2119 (2013)
25. Larsson, E., Shcherbakov, V., Heryudono, A.: A least squares radial basis function partition of unity method for solving PDEs. SIAM J. Sci. Comput. **39**, A2538–A2563 (2017)
26. Lazzaro, D., Montefusco, L.: Radial basis functions for the multivariate interpolation of large scattered data sets. J. Comput. Appl. Math. **140**, 521–536 (2002)
27. Melenk, J.M., Babuška, I.: The partition of unity finite element method: Basic theory and applications. Comput. Methods. Appl. Mech. Eng. **139**, 289–314 (1996)
28. Mollapourasl, R., Fereshtian, A., Li, H., Lu, X.: RBF-PU method for pricing options under the jump-diffusion model with local volatility. J. Comput. Appl. Math. **337**, 98–118 (2018)
29. Renka, R., Brown, R.: Algorithm 792: Accuracy tests of ACM algorithms for interpolation of scattered data in the plane. ACM Trans. Math. Softw. **25**, 78–94 (1999)
30. Rippa, S.: An algorithm for selecting a good value for the parameter c in radial basis function interpolation. Adv. Comput. Math. **11**, 193–210 (1999)
31. Scheuerer, M.: An alternative procedure for selecting a good value for the parameter c in RBF-interpolation. Adv. Comput. Math. **34**, 105–126 (2011)
32. Scheuerer, M., Schaback, R., Schlather, M.: Interpolation of spatial data: a stochastic or a deterministic problem? Eur. J. Appl. Math. **24**, 601–629 (2013)
33. Shepard, D.: A two-dimensional interpolation function for irregularly-spaced data. In: ACM '68: Proceedings of the 1968 – 23rd ACM national conference, pp. 517–524 (1968)
34. Uddin, M., Ali, H., Taufiq, M.: On the approximation of a nonlinear biological population model using localized radial basis function method. Math. Comput. Appl. **24**, 54 (2019)
35. Wendland, H.: Fast evaluation of radial basis functions: methods based on partition of unity. In: C.K. Chui, L.L. Schumaker, J. Stöckler (eds.) Approximation Theory X: Wavelets, Splines, and Applications, pp. 473–483. Vanderbilt University Press (2002)
36. Wendland, H.: Scattered Data Approximation, Cambridge Monographs on Applied and Computational Mathematics, vol. 17. Cambridge University Press, Cambridge (2005)
37. Wong, R., Luk, W., Heng, P.: Sampling with Hammersley and Halton points. J. Graph. Tools **2**, 9–24 (1997)
38. Zhang, Q., Zhao, Y., Levesley, J.: Adaptive radial basis function interpolation using an error indicator. Numer. Algorithms **76**, 441–471 (2017)