Journal of Computational and Applied Mathematics 451 (2024) 116108

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Parameter tuning in the radial kernel-based partition of unity method by Bayesian optimization

Roberto Cavoretto[1], Alessandra De Rossi[1], Sandro Lancellotti *,[1], Federico Romaniello

Department of Mathematics "Giuseppe Peano", University of Torino, via Carlo Alberto 10, 10123 Torino, Italy

A R T I C L E I N F O

Keywords:
Partition of unity interpolation
Radial basis functions
Kernel-based methods
Hyperparameter search
Bayesian optimization

A B S T R A C T

In this paper, we employ Bayesian optimization to concurrently explore the optimal values for both the shape parameter and the radius in the partition of unity interpolation using radial basis functions. Bayesian optimization is a probabilistic, iterative approach that models the error function through a progressively self-updated Gaussian process. Meanwhile, the partition of unity approach harnesses a meshfree method, allowing us to significantly reduce computational expenses, particularly when considering a substantial number of scattered data points. This reduction in computational cost is achieved by decomposing the entire domain into several smaller subdomains, each of them with a variable radius. We provide an estimation of the complexity of our algorithm and carry out numerical experiments to illustrate the effectiveness of our approach, dealing with test and real-world datasets.

1. Introduction

Over the past few decades, radial basis function (RBF) approximation and interpolation have emerged as a dynamic and significant tool for advancing meshfree techniques in the solution of various types of scientific and engineering problems, see e.g. [1–5]. They offer several advantageous features, such as straightforward implementation in higher dimensions, adaptability to various geometric configurations, and reasonable convergence properties, to name a few [6]. However, they may lead to a full, computationally expensive and ill-conditioned linear system. To overcome this drawback, in this work we focus on a meshfree method, known as the RBF partition of unity method (RBF-PUM), which makes use of local RBF approximants accumulating all the local contributions in a global partition of unity fit. A first version of PUM is introduced in [7] to reconstruct a function from scattered data points. This approach hinges on the concept of localizing the approximation process by a decomposition of the original big problem into several small subproblems, thus finding application in many fields of computational mathematics and scientific computing [8–11]. Indeed, the PUM is used to efficiently split the data within smaller subdomains or balls. The first combination of the PUM with the RBF interpolation goes back to [12], where an error analysis is also given for functions in the native space of the underlying RBFs. The RBF-PUM proposed in this paper is obtained by a weighted sum of local RBF interpolants depending on a shape (or scale) parameter ε and a variable radius δ in each subdomain.

Moreover, in this study we employ a well-studied statistical method known as *Bayesian Optimization* (BO) [13] to simultaneously search for the optimal values of (ε, δ) within each subdomain of RBF-PUM. The BO, originally developed in the field of machine learning for optimizing complex or hard-to-assess functions, finds utility in hyperparameter tuning tasks by circumventing the need

* Corresponding author.
E-mail addresses: roberto.cavoretto@unito.it (R. Cavoretto), alessandra.derossi@unito.it (A. De Rossi), sandro.lancellotti@unito.it (S. Lancellotti), federico.romaniello@unito.it (F. Romaniello).
[1] Member of the INdAM Research group GNCS.

https://doi.org/10.1016/j.cam.2024.116108
Received 11 October 2023; Received in revised form 3 May 2024
Available online 22 June 2024

to compute and evaluate the approximations for parameter combinations that are far from optimal. All the algorithms involved in the main procedure are described and analyzed in detail in order to show how this approach leads to a substantial reduction in terms of computational time. Numerical experiments on some benchmark test cases and real-world datasets such as Tonga Trench and Franke's glacier ones point out that also in applied contexts a good accuracy of the interpolant is preserved.

The paper is organized as follows. In Section 2, the RBF-PUM interpolation problem is stated. In Section 3, BO Gaussian processes and acquisition functions are presented. Section 4 contains a description of the algorithms and their complexity analysis. In Section 5 numerical experiments show the efficacy of our scheme by solving interpolation problems on some test examples and real-world applications. Section 6 concludes the paper.

2. RBF-PUM interpolation

In this section, we introduce the interpolation problem and the basic theory on RBF-PUM, highlighting the reasons that inspired this paper.

2.1. The RBF method

Let $X = \{\boldsymbol{x}_i, i = 1, \dots, N\}$ be a set of distinct data points or nodes arbitrarily distributed on a domain $\Omega \subseteq \mathbb{R}^d$. Associated with this set is another collection $F = \{f_i = f(\boldsymbol{x}_i), i = 1, \dots, N\}$ representing data values obtained by sampling a potentially unknown function $f : \Omega \to \mathbb{R}$ at the nodes $\boldsymbol{x}_i$. The problem at hand is the *scattered data interpolation problem*, which entails discovering an interpolating function $P_f : \Omega \to \mathbb{R}$ that exactly reproduces the measured values at their respective locations, i.e.

$$P_f\left(\boldsymbol{x}_i\right) = f_i, \qquad i = 1, \dots, N.$$

We now suppose to have a univariate function $\varphi : [0, \infty) \to \mathbb{R}$, known as RBF, which depends on a shape parameter $\varepsilon > 0$ providing, for $\boldsymbol{x}, \boldsymbol{z} \in \Omega$, the real symmetric strictly positive definite kernel [14]

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

The kernel-based interpolant P_f can be written as

$$P_f(\boldsymbol{x}) = \sum_{k=1}^{N} c_k \kappa_\varepsilon\left(\boldsymbol{x}, \boldsymbol{x}_k\right), \quad \boldsymbol{x} \in \Omega,$$

whose coefficients c_k are the solution of the linear system

$$\mathsf{K}\boldsymbol{c} = \boldsymbol{f}, \tag{1}$$

where $\boldsymbol{c} = \left(c_1, \dots, c_N\right)^\intercal$, $\boldsymbol{f} = \left(f_1, \dots, f_N\right)^\intercal$, and $\mathsf{K}_{ik} = \kappa_\varepsilon\left(\boldsymbol{x}_i, \boldsymbol{x}_k\right)$, $i, k = 1, \dots, N$. Since κ_ε is a symmetric and strictly positive definite kernel, the system (1) has exactly one solution [15]. Furthermore, the kernel κ_ε gives rise to what is known as the *native space*. This native space, denoted as $\mathcal{N}\kappa_\varepsilon(\Omega)$, is a Hilbert space equipped with the inner product $(\cdot,\cdot)_{\mathcal{N}\kappa_\varepsilon(\Omega)}$. In this space, the kernel κ_ε is reproducing, i.e. for any $f \in \mathcal{N}_{\kappa_\varepsilon}(\Omega)$ the following identity holds: $f(\boldsymbol{x}) = (f, \kappa_\varepsilon(\cdot, \boldsymbol{x}))_{\mathcal{N}_{\kappa_\varepsilon}(\Omega)}$, with $\boldsymbol{x} \in \Omega$. By introducing a pre-Hilbert space $H_{\kappa_\varepsilon}(\Omega) = \text{span}\{\kappa_\varepsilon(\cdot, \boldsymbol{x}),\ \boldsymbol{x} \in \Omega\}$, with reproducing kernel κ_ε and equipped with the bilinear form $(\cdot,\cdot)_{H_{\kappa_\varepsilon}(\Omega)}$, the native space $\mathcal{N}_{\kappa_\varepsilon}(\Omega)$ of κ_ε coincides with its completion with respect to the norm $\|\cdot\|_{H_{\kappa_\varepsilon}(\Omega)} = \sqrt{(\cdot,\cdot)_{H_{\kappa_\varepsilon}(\Omega)}}$, and for all $f \in H_{\kappa_\varepsilon}(\Omega)$ we have $\|f\|_{\mathcal{N}_{\kappa_\varepsilon}(\Omega)} = \|f\|_{H_{\kappa_\varepsilon}(\Omega)}$.

2.2. The PUM scheme

It is a widely recognized fact that performing the inversion of the kernel interpolation matrix in (1) can become computationally demanding as the amount of data significantly grows. To address this challenge effectively, a practical approach is to divide the open and bounded domain Ω into m overlapping subdomains denoted as Ω_j, with the property that $\Omega \subseteq \bigcup_{j=1}^{m} \Omega_j$. Consequently, this allows for the problem of interpolation to be split independently within each of these subdomains.

The PU covering consists of overlapping balls of radius δ whose centers are the grid poimts $P = \{\tilde{\boldsymbol{x}}_k,\ k = 1, \dots, m\}$. In [3] it is shown that when the nodes are nearly uniformed distributed, m is a suitable number of PU subdomains on Ω if $N/m \approx 2^d$. Then, the covering property is satisfied by taking the radius δ such that

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

The PUM solves a local interpolation problem on each subdomain and constructs the global approximant by gluing together the local contributions using weights. To achieve that, we need those weights to be a family of compactly supported, non-negative, continuous functions w_j, with $\text{supp}\left(w_j\right) \subseteq \Omega_j$, such that

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

Once we choose the partition of unity $\{w_j\}_{j=1}^m$, the global interpolant is formed by the weighted sum of m local approximants P_f^j, i.e.

$$P_f(\boldsymbol{x}) = \sum_{j=1}^{m} P_f^j(\boldsymbol{x})\, w_j(\boldsymbol{x}) = \sum_{j=1}^{m} \left(\sum_{k=1}^{N_j} c_k^j \kappa_{\varepsilon,\delta}(\boldsymbol{x}, \boldsymbol{x}_k^j) \right) w_j(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega,$$

where $\boldsymbol{x}_k^j \in X_j = X \cap \Omega_j$ and $N_j = |X_j|$, with $k = 1, \ldots, N_j$. We will use the following and well-known Shepard weights in the implementation of our algorithms:

$$w_j(x) = \frac{\varphi_j(x)}{\sum_{k=1}^{m} \varphi_k(x)} \quad j = 1, \ldots, m, \tag{2}$$

where $\varphi_k(x)$ is a compactly supported function with support on Ω_j, see [7,16].

It is worth noting that the accuracy of the fit strongly depends on the choices of the shape parameter and the radius, see e.g. [17–23].

An advantage of this scheme is the use of a continuous search in the parameters space $\mathcal{X} = I \times J = (0, \varepsilon_{max}] \times [\delta_{min}, 2\delta_{min}]$ to obtain a better approximation of $(\varepsilon, \delta)_j^*$, in each subdomain Ω_j.

3. Bayesian optimization

When seeking to locate a global maximizer for an unknown or challenging-to-assess function g within a bounded set X, Bayesian optimization offers an effective approach [24]. Highly regarded in the realm of machine learning, BO is an iterative methodology that optimally utilizes available resources. It entails constructing a probabilistic model of g, often referred to as a *surrogate model*, and employing it to guide the selection of sampling points within the set X using an acquisition function. These selected points are located in the area in which the target function will be assessed. After each iteration, the distribution is updated to reflect the acquired information and is subsequently utilized in the next iteration. While some computational effort is required to determine the next point for evaluation, this cost is justifiable when the evaluations of g are computationally expensive. This is because such computations are driven by the goal of reaching the maximum value in a limited number of iterations, which is particularly important in scenarios like optimizing the error function of resource-intensive machine learning algorithms in multi-layer neural networks.

Hereinafter, we briefly review the BO technique [25]. A Gaussian Process (GP) is a collection of random variables such that any subsets of these have a joint Gaussian distribution. Then GPs are completely specified by a mean function $\mathtt{m} : \mathcal{X} \to \mathbb{R}$ and a positive definite covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ (see [26]). Further, they are the most common choice for the surrogate model for BO due to the low evaluation cost and the ability to incorporate prior beliefs about the objective function. When modeling the target function with a GP as $g(\mathbf{x}) \sim \mathcal{GP}\big(\mathtt{m}(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big)$, we impose that

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

In the matter of making a prediction given by some observations, the assumption of joint Gaussianity allows retrieving the prediction using the standard formula for mean and variance of a conditional normal distribution. Hence, suppose to have s observation $\boldsymbol{g} = (g(\mathbf{x}_1), \ldots, g(\mathbf{x}_s))^\intercal$ on $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_s)^\intercal$ and a new point $\bar{\mathbf{x}}$ on which we are interested in having a prediction of $\bar{g} = g(\bar{\mathbf{x}})$. The previous observations $\boldsymbol{g}$ and the predicted value $g(\bar{\mathbf{x}})$ are jointly normally distributed:

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

where $K(\mathbf{X}, \mathbf{X})$ is the $s \times s$ matrix with (i, j)-element $k(\mathbf{x}_i, \mathbf{x}_j)$, and $K(\mathbf{X}, \bar{\mathbf{x}})$ is a $s \times 1$ vector whose ith element is given by $k(\mathbf{x}_i, \bar{\mathbf{x}})$, see [26]. Since $Pr(f(\bar{\mathbf{x}})|\boldsymbol{g})$ must also be normal, it is also possible to estimate the distribution, the mean and the covariance, for any point in the domain. When data points and data values retrieved by the evaluation of the target function are fed to the model, they induce a posterior distribution over functions which is used for the next iteration as a prior. It is worth noting that in the case of modeling a function with a GP, when we observe a value, we are essentially observing the random variable associated with that specific point.

An *acquisition function* $a : \mathcal{X} \to \mathbb{R}$ serves as a tool for determining the subsequent point at which the objective function will be assessed. The goal is to choose a point that maximizes this acquisition function, and the result of evaluating the objective function at this chosen point is utilized to update the surrogate model. The design of an acquisition function is specifically crafted so that a high acquisition score corresponds to the likelihood of encountering high values of the objective function. When the decision is made regarding which acquisition function to use, a trade-off arises between exploration and exploitation. Exploration involves the selection of points characterized by high levels of uncertainty, typically those located at a considerable distance from previously examined points. Conversely, exploitation involves the selection of points in close proximity to those already assessed by the objective function. The most common acquisition functions are:

- **Probability of Improvement**, which maximizes the probability of improvement over the best current value;
- **Expected Improvement**, which maximizes the expected improvement over the current best;

- **GP Upper Confidence Bound**, which minimizes the cumulative regret.[2]

The "Expected Improvement" [27] acquisition function not only considers the probability of improvement of the candidate point with respect to the previous maximum, but also the magnitude of this improvement. Suppose that after a number of iterations the current maximum of the objective function is $g(\hat{\mathbf{x}})$. Given a new point $\mathbf{x}$, the Expected Improvement acquisition function computes the expectation of improvement $g(\mathbf{x}) - g(\hat{\mathbf{x}})$ over the part of the normal distribution that is above the current maximum:

$$EI(\mathbf{x}) = \int_{g(\hat{\mathbf{x}})}^{\infty} \left(g^*(\mathbf{x}) - g(\hat{\mathbf{x}})\right) \frac{1}{\sqrt{2\pi}\sigma(\mathbf{x})} e^{-\frac{1}{2}[(g^*(\mathbf{x})-\mu(\mathbf{x}))/\sigma(\mathbf{x})]^2} dg^*(\mathbf{x}), \tag{3}$$

where $g^*(\mathbf{x})$, $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ represent the predicted value by the surrogate model, the expected value and the variance of $\mathbf{x}$, respectively. Solving integral (3) leads to the following closed form for the evaluation of the Expected Improvement:

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

where $Z = \frac{\mu(\mathbf{x})-g(\hat{\mathbf{x}})}{\sigma(\mathbf{x})}$, while ϕ and Φ are the Probability Density Function and Cumulative Distribution Function of the standard normal distribution $\mathcal{N}(0,1)$, respectively.

To perform the experiments we used an extension of , proposed in [28], that also trades off exploration and expectation by means of a non-negative parameter ξ:

$$EI(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - g(\hat{\mathbf{x}}) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z), & \text{if } \sigma(\mathbf{x}) > 0, \\ 0, & \text{if } \sigma(\mathbf{x}) = 0, \end{cases} \tag{4}$$

where $Z = \frac{\mu(\mathbf{x})-g(\hat{\mathbf{x}})-\xi}{\sigma(\mathbf{x})}$.

4. Algorithms and their computational cost

In this section, we first describe in Section 4.1 the algorithms for interpolation processes with Bayesian optimization. In Section 4.2 their computational cost is analyzed.

4.1. Algorithms

Let X be a set of points for which we know the associated set of data values F, and let $\bar{X} = \{\bar{\mathbf{x}}_i, i = 1,\ldots,\bar{n}\}$ be a set of points on which we want to evaluate some approximate solutions. The whole process is handled by Algorithm 1, which invokes the Bayesian optimization (Algorithm 3) for the parameters search and the partition of unity (Algorithm 5) for the evaluation of the approximant. In detail, Algorithm 1 builds the approximant on $\bar{X}$ with the best shape parameters $\boldsymbol{\varepsilon}$ and radii $\boldsymbol{\delta}$ found by means of the Bayesian optimization. With the aim of doing that, the algorithm starts by retrieving N, the number of points in X, and the dimension d of the space. Using these values, it evaluates the number m of partition of unity centers and generates them as an equally spaced grid in Ω. For the sake of clarity, hereinafter, without loss of generality we consider the special case of $\Omega = [0,1]^d$. We remark that a suitable number of subdomains is $\lfloor \frac{N}{2^d} \rfloor$, see [3]. After that, it evaluates the distance tree of X, the function *KDTree* of the package *scipy.spatial* and its method are used to build and perform the points search on it. Next, in each subdomain the value for the radius that ensures the minimum density is found by Algorithm 2. This value δ_{start} will be used when applying a Bayesian optimization (Algorithm 3) to enhance the shape parameter and the radius in each subdomain. The last step is to train the RBF-PUM approximant (Algorithm 4) with the found parameters and return the approximated value of the function on the set of points $\bar{X}$.

The core of the process is accomplished by Algorithm 3, which is a remodeling of the *BayesianOptimization* Python's library [29]. Concretely, it traces the optimization process provided by the *optimization* method of the *BayesianOptimisation* class, which in sequence uses the methods *fit* and *predict* of the function *GaussianProcessRegressor* of the *sklearn.gaussian_process* package [30]. Further details about the implementation of *GaussianProcessRegressor* are available at [26, Algorithm 2.1].

To measure the goodness of the approximant, we introduce the Maximum Absolute Error (MAE), the Relative Maximum Absolute Error (RMAE) and the Relative Root Mean Squared Error (RRMSE) defined as follows:

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

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

[2] 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 $g(x^*) - g(x)$ over the course of the optimization.

Algorithm 1 BO-PUM

```
  Input:
    $X$: data points, $F$: data values, $\bar{X}$: evaluation points, $I$: $\varepsilon$ search interval, $a$: acquisition function, $\xi$: exploration–exploitation parameter, $nstart$: number of starting points, $niter$: number of Bayesian iterations, $min_{pts}$: minimum number of points in a subdomain, $\tau$: tolerance of the error during the parameters search, $w$: weight function.

$N \to$ number of points in $X$
$d \to$ space dimension of $X$
$m \to \lfloor \frac{N}{2^d} \rfloor$
$centers \to$ grid of $m$ points in $\Omega = [0,1]^d$
$\boldsymbol{\varepsilon} \to$ 0-vector of length $m$
$\boldsymbol{\delta} \to$ 0-vector of length $m$
$DT_X \to$ distance tree of $X$
$\boldsymbol{\delta}_{start} \to$ FIND-MIN-RADIUS$(X, centers, m, d, min_{pts}, DT_X)$
for $i = 1 : |centers|$ do
    $J \to [\boldsymbol{\delta}_{start}[i], 2\boldsymbol{\delta}_{start}[i]]$
    $[\varepsilon, \delta] \to$ BO$(X, F, I, J, a, \xi, nstart, niter, DT_X, centers[i], \tau)$
    $\boldsymbol{\varepsilon}[i] \to \varepsilon$
    $\boldsymbol{\sigma}[i] \to \sigma$
end for
$\mathbf{P}_f \to$ PUM$(X, F, \bar{X}, centers, DT_X, w, \boldsymbol{\varepsilon}, \boldsymbol{\delta})$
  Output:
    $\mathbf{P}_f$: evaluation of the interpolated solution on $\bar{X}$
```

Algorithm 2 FIND-MIN-RADIUS

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

$\boldsymbol{\delta}_{start} \to \frac{\sqrt{d}}{2m^{\frac{1}{d}}} \times$ 1-vector of length $|centers|$
for $i = 1 : m$ do
    $X_{sub} \to$ retrieve the subset of $X$ within distance $\boldsymbol{\delta}_{start}[i]$ from $centers[i]$ using $DT_X$
    while $|X_{sub}| < min_{pts}$ do
        $\boldsymbol{\delta}_{start}[i] \to \boldsymbol{\delta}_{start}[i] + \frac{1}{8}\frac{\sqrt{d}}{2m}$
        $X_{sub} \to$ retrieve the subset of $X$ within distance $\boldsymbol{\delta}_{start}[i]$ from $centers[i]$ using $DT_X$
    end while
end for
  Output:
    $\boldsymbol{\delta}_{start}$: vector of subdomain radii that ensure the minimum densities.
```

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

where X and F are the given sets of data points and data values and $\mathbf{P}_f = (P_f(\boldsymbol{x}_1), \dots P_f(\boldsymbol{x}_N))$, with $N = |X|$. The first step is to initialize the function g to optimize, defined as the negative $\mathrm{MAE}_{X_{val},F_{val}}(\cdot)$ between the known values F_{val} and the approximation evaluated by Algorithm 4 for the points X_{val} for a specific value of θ, and the search space $\mathcal{X}$. We remark that X_{val}, F_{val} are subsets of X used for the BO search. Then, until the number of iterations is reached or the error drops below a certain prescribed tolerance, for the first $nstart$ iteration the algorithm randomly samples $\hat{\theta} = (\hat{\varepsilon}, \hat{\delta})$ in the search domain $\mathcal{X}$, otherwise the chosen point $\hat{\theta}$ is the one that maximizes the acquisition function (4) evaluated on a random set (the acquisition function exploits the Gaussian process fitted in the previous iteration). At this point the algorithm retrieves the subsets $X_j \subset X$ and $\tilde{X}_j \subset \bar{X}$ that are contained in the related subdomain Ω_j, splits them into training and validation subsets and fits an approximant applying Algorithm 4. At each iteration the values of $\hat{\theta}$ and the error obtained fitting the interpolant with the parameter $\hat{\theta}$ are stored in the vectors $\boldsymbol{\theta}$ and $\boldsymbol{g}$ and a Gaussian process on $(\boldsymbol{\theta}, \boldsymbol{g})$ is fitted. The last step consists of determining θ^*, the parameter that maximizes the vector $\boldsymbol{g}$.

The fundamental component of the scheme is represented by Algorithm 4, which solves the interpolation system, and finds the approximated values of the function on $\bar{X}$.

Algorithm 3 BO

```
  Input:
    $X$: data points, $F$: data values, $I$: parameter search interval for $\varepsilon$, $J$: parameter search interval for $\delta$, $a$: acquisition function,
    $\xi$: exploration–exploitation parameter, $nstart$: number of starting points, $niter$: number of Bayesian iterations, $DT_X$: distance
    tree of $X$, $center$: subdomain center, $\tau$: tolerance.

Set $g \to -\mathrm{MAE}_{X_{val},F_{val}}(\cdot)$
$\mathcal{X} \to I \times J$
$\boldsymbol{g} \to (\cdot)$ (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
    $X_{sub} \to$ retrieve the subset of $X$ within distance $\hat{\delta}$ from $center$ using $DT_X$
    Split $X_{sub}, F_{sub}$ in $X_{train}, X_{val}, F_{train}, F_{val}$
    $\mathbf{P}_{f_{\hat{\theta}}} \to \mathbf{RBF}(X_{train}, F_{train}, X_{val}, \hat{\varepsilon})$    (call to Algorithm 4)
    $\boldsymbol{\theta} \to \boldsymbol{\theta} \cup \hat{\theta}$
    $\boldsymbol{g} \to \boldsymbol{g} \cup g(\mathbf{P}_{f_{\hat{\theta}}})$
    Fit the Gaussian process on $(\boldsymbol{\theta}, \boldsymbol{g})$
    $i \to i + 1$
end while
$\theta^* \to \operatorname{argmax} \boldsymbol{g}$
  Output:
    $\theta^*$: best parameters.
```

Algorithm 4 RBF

```
  Input:
    $X$: data points, $F$: data values, $\bar{X}$: evaluation points, $\varepsilon$: shape parameter.

$\mathbf{P}_f \to$ Solve the linear system of the form (1)
  Output:
    $\mathbf{P}_f$: evaluation of the interpolated solution on $\bar{X}$.
```

Algorithm 5, given the values of ε and δ for each subdomain, deals with determining local solutions and summing them up to construct a global one. In detail, it determines the Shepard weights in (2) for the subdomains and the approximate evaluation array on $\bar{X}$ is initialized with all zeros.

It is worth remarking that the algorithms presented in this section can be extended to manage the approximation settings by selecting a subset of X as the set of centers for the RBFs and solving a linear system of the form (1) in the least squares setting, as done in [31]. This approach is particularly compelling in scenarios with a high volume of data points or when the dataset is noisy. In such cases, opting for an approximation technique over interpolation is preferable to avoid interpolating the noise. We refer the reader to [15] for further details.

4.2. Computational analysis of algorithms

In this section we discuss the complexity of the Algorithm 1 by first analyzing its constituent components. We remark here that logarithms are taken base 2. Without explicitly declaring the shape parameter and the subdomain radius, we will show that the total expense required to build a global interpolant is $\mathcal{O}(N^{\frac{2d+1}{d}} + N(nstart+niter)(N + N_j^3 + (nstart+niter)^3))$, where $N = |X|$, d is the space dimension, N_j is the maximum among the number of points in the subdomains (note that $N_j \ll N$), and $nstart$ and $niter$ are the initialization and the Bayesian steps of the optimization. We will need to consider the cost of construction and search in a kdtree, $\mathcal{O}(N\log(N))$ and $\mathcal{O}(N)$, respectively. We use a kdtree implementation by *scipy* [32] that provides, for a balanced dataset, a balanced tree by applying a median-based splitting strategy in $\mathcal{O}(N\log(N))$. We want to highlight that hardly ever and very uncommon in practice, the computational expenses for the construction could be $\mathcal{O}(N^2)$ in the worst case. A different discussion can be addressed in the case of search, where we face three different scenarios: the best-case scenario, where for balanced tree, the search cost $\log(N)$;

Algorithm 5 PUM

```
  Input:
    $X$: data points, $F$: data values, $\bar{X}$: evaluation points, $centers$: subdomain centers, $DT_X$: distance tree of $X$, $w$: weight function,
    $\varepsilon$: vector of shape parameters, $\delta$: vector of subdomain radius.

sw $\rightarrow$ retrieve Shepard weights from $w$
$\mathbf{P}_f \rightarrow$ 0-vector of length $|\bar{X}|$
$DT_{\bar{X}} \rightarrow$ distance tree of $\bar{X}$
for $j = 1 : |centers|$ do
  $n_j \rightarrow$ indices of points of $X$ at distance $\delta[j]$ from $centers[j]$ using $DT_X$
  if $|n_j| \neq 0$ then
    $s_j \rightarrow$ points of $\bar{X}$ at distance $\delta[j]$ from $centers[j]$ using $DT_{\bar{X}}$
    if $(|s_j| \neq 0)$ then
      $\mathbf{P}'_{f_{\mathbf{s_j}}} \rightarrow \mathbf{RBF}(X_j, F_j, \bar{X}_{s_j}, \varepsilon_j)$
      $\mathbf{P}_{f_{\mathbf{s_j}}} \rightarrow \mathbf{P}_{f_{\mathbf{s_j}}} + \mathbf{P}'_{f_{\mathbf{s_j}}} * \mathbf{sw}_{\mathbf{s_j}}$
    end if
  end if
end for
  Output:
    $\mathbf{P}_f$: evaluation of the interpolated solution on $\bar{X}$.
```

the average-case, where the kdtree is reasonably balanced and the search cost is $\mathcal{O}(N+K)$ where K is the number of points found in the search distance; the worst-case, where the tree is unbalanced and the computational cost is $\mathcal{O}(N)$.

FIND-MIN-RADIUS: Suppose that the initial radius for a subdomain is equal to 0. In this case, the maximum number of augmenting steps inside the *while* loop is bounded by the length of the diagonal of the d-dimensional hypercube over the weight of the extent. Hence it is bounded with $16m^{\frac{1}{d}} \simeq 8N^{\frac{1}{d}}$. Taking into account that the search has a cost of N, we can estimate the cost of the whole radius search in Algorithm 2 as:

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

RBF: Algorithm 4 simply involves the solution of a linear system. With an input of N nodes the computational expense is $\mathcal{O}(N^3)$.

PUM: Overlooking the cost of the computation of the Shepard weights and some initialization that has cost linearly dependent from N, we have a *for* loop of length m where for each iteration we have three point search of cost N, a call of Algorithm 4 with a variable input dimension and an update of a N-length vector. Let $\tilde{N} = \max_j N_j = \max_j |X_j|$ be, i.e. it is the maximum among the number of points in the subdomain. We have that the complexity of Algorithm 5 is:

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

BO: Algorithm 3 is made by a *while* loop of length less than $nstart+niter$ iteration. For each iteration the cost can be summarized as follows: computation of order N for selecting the next parameter to evaluate, 2 points search, a splitting of cost N, an invocation of Algorithm 4 of cost $\tilde{N}^3$ and the fitting of the Gaussian process that in the worst case cost $(nstart + niter)^3$. The total expense for the Bayesian optimization is:

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

BO-PUM: In conclusion, summing up the previous results, and taking into account that the construction of the kdtrees requires computation of order $N\log(N)$, we can retrieve the computational expense for Algorithm 1 as follows:

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

5. Numerical experiments and applications

In this section, we will illustrate how algorithms presented in Section 4 work efficiently both on test and real-world datasets.

Before going into details, we remark that all the code was developed in Python 3.9 and the library used to perform the optimization is *BayesianOptimization* [29] in which the default kernel used for the Gaussian process is the Matérn 5/2. Moreover, we set the parameters $\xi = 0.15$ and $min_{pts}= 15$. To apply BO in the search for optimal parameters (ε, δ), we assume that the objective function to be maximized is the Maximum Absolute Error (MAE) of the RBF interpolant, with the sign inverted. This is because BO is a maximization process, as explained in Section 3. We vary the number of points in the training set while keeping the test set fixed at 1000 points. During the BO process, for each subdomain, after identifying the points within it, we further divide them into sub-training and sub-validation sets to enable the evaluation of the training error. After determining the best parameter pairs for each subdomain, we train a PUM interpolant on the training set for each optimizer using the identified parameters. For each subdomain, the search space is $\mathcal{X} = (0, 20] \times [\delta_{min}, 2\delta_{min}]$, where δ_{min} is the radius value that ensures a minimum density in the subdomain. BO performs 5 random steps plus at most 25 Bayesian steps in the search space. The iterative process stops when the desired tolerance τ is reached. Notice that setting the tolerance to the machine precision is equivalent to forcing the algorithm to exhaust all iterations.

5.1. Numerical experiments

We perform the experiments on four different sizes of random data in the domain $\Omega = [0,1]^2$ using three RBFs of different smoothness, i.e.,

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

and the following test functions [33,34]:

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

Results are shown in Tables 1–3. It is worth noting that as the number of points increases, the execution time of the BO decreases. This is due to the high density of the space when a greater number of points is considered. In particular, when this happens, there are denser subdomains, and thus better accuracy and fewer BO iterations are needed to satisfy the tolerance τ.

To test the goodness of the prediction, we conduct a Mann–Whitney U test [35], where the null hypothesis is that the probability distribution of a randomly drawn observation from one group is the same as the probability distribution of a randomly drawn observation from the other group, against the alternative hypothesis that these distributions are not equal. In our case, the first sample is made by test data, while the second is made by the approximated values. To perform the test we used the function *mannwhitneyu* of the library *scipy* [32] obtaining, for all the experiments, a p-value greater than 0.05, indicating that we cannot reject the null hypothesis. Therefore, we conclude that the samples belong to the same distribution.

To highlight the importance of the search phase for the parameters we report in Table 4 the results obtained by picking $\varepsilon = 10$, that is midpoint of the search interval for ε, and $\delta = \sqrt{2/N}$, namely the minimum radius that ensures the covering property. As shown by the results, choosing parameters without a criterion leads to unsatisfactory results.

Remark 1. As an example we compute the local condition numbers for each subdomain after the optimization on $N = 5000$ points with the function f_2 and the kernel φ_2, obtaining an average and maximum condition number of $5.15\text{e}+14$ and $3.44\text{e}+17$, respectively. To handle large condition numbers, we may suggest a few normalization techniques such as the truncated SVD and the Tikhonov regularization and randomized GSVD in [36–38].

All tests are carried out on a MacBook Air (2020), 1.2 GHz Quad-Core Intel Core i7 processor, 16 GB 3733 MHz LPDDR4X RAM, via Python 3.9.12.

5.2. Real data applications

In this subsection we show the behavior of our framework PUM-BO applied on two different real data examples showing the performance of the algorithm when the measurements are taken with regular intervals, similar to grid points, and on contour lines, similar to random measurements.

Tonga Trench Dataset: The Tonga Trench, situated within the vast expanse of the Pacific Ocean, descends to an astonishing depth of 10,882 meters (35,702 ft) at its lowest point, aptly named Horizon Deep. This trench, accompanied by an adjacent volcanic island arc, constitutes an active subduction zone nestled between two tectonic plates within Earth's lithosphere.

In our example we consider a dataset consisting of 8113 points. We split it in a training and test set of 7000 and 1113 random samples without repetition, shuffling the indices and selecting the first 7000 for the training set and the remainder for the test set (see Fig. 1) (see Table 5).

Table 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.

N	τ	Gaussian kernel (φ_1)		Matérn kernel (φ_2)	
		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 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.

N	τ	Gaussian kernel (φ_1)		Matérn kernel (φ_2)	
		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 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	τ	f_1		f_2	
		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

Franke's Glacier Dataset: This dataset previously used in [39] for interpolation of scattered data for surface fitting, consists of 8338 measurements of altitude of a glacier. Unfortunately we cannot find any background on where these data were collected or indeed even the location of this glacier. More details on this dataset can be found at https://search.r-project.org/CRAN/refmans/fields/html/glacier.html. However, it is an interesting dataset in which it appears that the elevations are reported along lines of equal elevation, i.e. contours, perhaps from a digitization of a topographic map or survey. In our example we consider the whole dataset consisting of 8338 points and we split it in a training and test set of 7000 and 1338 random samples without repetition, shuffling the indices and selecting the first 7000 for the training set and the remainder for the test set (see Fig. 2) (see Table 6).

6. Conclusions

In this paper, we employed BO for the simultaneous search of the subdomain radius and RBF shape parameter in the partition of unity algorithm. As a statistical technique, it effectively guides parameter selection towards those values that ensure a predetermined tolerance. Across all presented examples, BO consistently achieves the required accuracy. The differentiating factor lies in the search times, which, in some cases, decrease with an increasing number of points considered. This occurs because a larger number of points and denser distribution within each subdomain lead to reaching the required tolerance in fewer iterations, thus resulting in reduced processing time. Given the strong interest coming from the research community, in a future work we plan to extend the method to solving partial differential equations, also analyzing related computational and implementation details.

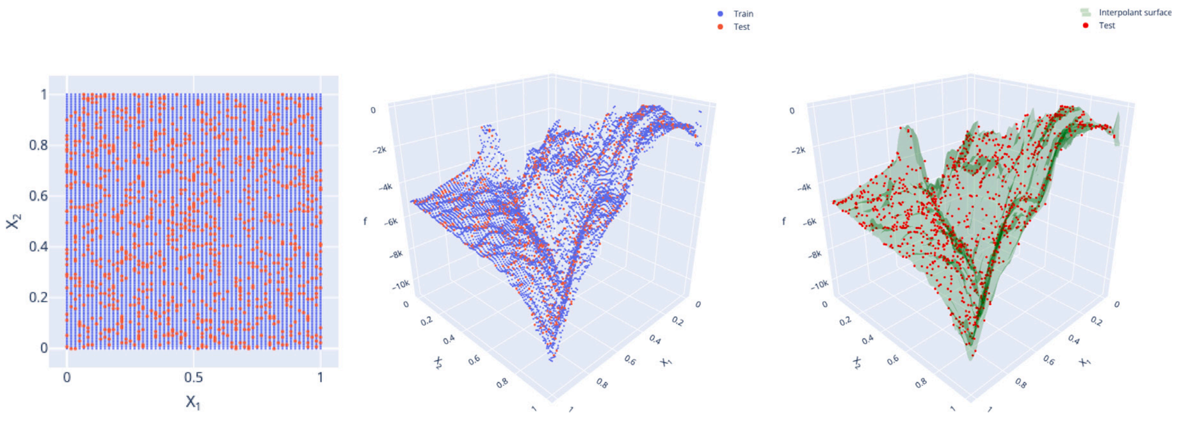

Fig. 1. Tonga Trench dataset: plain projection (left), 3D view (center), interpolating surface constructed on a 100×100 point grid in $[0,1]^2$.

Table 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 $\sqrt{2/N}$.

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
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.

τ	Gaussian kernel (φ_1)			Matérn kernel (φ_2)		
	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 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.

τ	Gaussian kernel (φ_1)			Matérn kernel (φ_2)		
	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

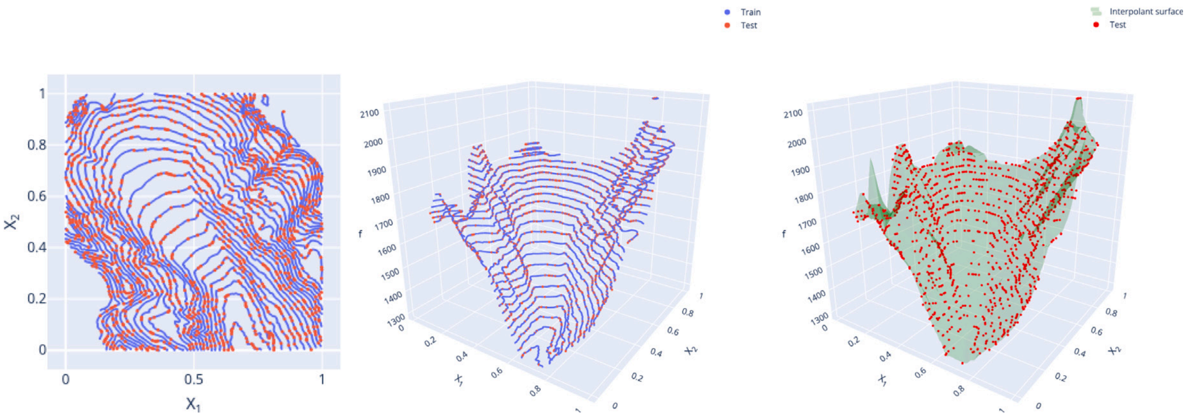

Fig. 2. Franke's glacier dataset: plain projection (left), 3D view (center), interpolating surface constructed on a 100 × 100 point grid in $[0,1]^2$.

Data availability

Data will be made available on request.

Acknowledgments

The authors sincerely thank the reviewers for their constructive and valuable comments that enabled to improve the paper. This research has been accomplished within the RITA "Research ITalian network on Approximation" and the UMI Group TAA "Approximation Theory and Applications". This work has been supported by the INdAM–GNCS 2022 Project "Computational methods for kernel-based approximation and its applications", code CUP_E55F22000270001, and by the Spoke "Future HPC & BigData" of the ICSC–National Research Center in "High-Performance Computing, Big Data and Quantum Computing", funded by European Union – NextGenerationEU. Moreover, the work has been supported by the Fondazione CRT, Italy, project 2022 "Modelli matematici e algoritmi predittivi di intelligenza artificiale per la mobilità sostenibile".

References

[1] M.E. Biancolini, Fast Radial Basis Functions for Engineering Applications, Springer Cham, 2018.
[2] 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.
[3] G.E. Fasshauer, Meshfree Approximation Methods with MATLAB, World Scientific, Singapore, 2007.
[4] E. Francomano, M. Paliaga, Highlighting numerical insights of an efficient SPH method, Appl. Math. Comput. 339 (2018) 899–915.
[5] R. Schaback, Small errors imply large evaluation instabilities, Adv. Comput. Math. 49 (2023) 25.
[6] H. Wendland, Scattered Data Approximation, in: Cambridge Monogr. Appl. Comput. Math., vol. 17, Cambridge Univ. Press, Cambridge, 2005.
[7] D. Shepard, A two-dimensional interpolation function for irregularly-spaced data, in: Proceedings of the 23rd National Conference ACM, 1968, pp. 517–523.
[8] I. Babuška, J.M. Melenk, The partition of unity method, Internat. J. Numer. Methods Engrg. 40 (4) (1997) 727–758.
[9] R. Cavoretto, Adaptive radial basis function partition of unity interpolation: A bivariate algorithm for unstructured data, J. Sci. Comput. 87 (2021) 41.
[10] R. Cavoretto, A. De Rossi, W. Erb, Partition of unity methods for signal processing on graphs, J. Fourier Anal. Appl. 27 (2021) 66.
[11] 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.
[12] 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, pp. 473–483.
[13] J. Snoek, H. Larochelle, R.P. Adams, Practical Bayesian optimization of machine learning algorithms, Adv. Neural Inf. Process. Syst. 25 (2012) 2960–2968.
[14] A. Iske, Scattered data approximation by positive definite kernel functions, Rend. Semin. Mat. Univ. Politec. Torino 69 (2011) 217–246.
[15] G.E. Fasshauer, M.J. McCourt, Kernel-Based Approximation Methods using MATLAB, World Scientific, Singapore, 2015.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] B. Fornberg, G. Wright, Stable computation of multiquadrics interpolants for all values of the shape parameter, Comput. Math. Appl. 47 (2004) 497–523.
[21] E. Larsson, B. Fornberg, Theoretical and computational aspects of multivariate interpolation with increasingly flat radial basis functions, Comput. Math. Appl. 49 (2005) 103–130.
[22] E. Larsson, R. Schaback, Scaling of radial basis functions, IMA J. Numer. Anal. (2023) drad035, http://dx.doi.org/10.1093/imanum/drad035.
[23] L. Ling, F. Marchetti, A stochastic extended Rippa's algorithm for LOOCV, Appl. Math. Lett. 129 (2022) 107955.
[24] J. Mockus, V. Tiesis, A. Zilinskas, The application of Bayesian methods for seeking the extremum, Towards Global Optim. 2 (1978) 117–129.

[25] 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.
[26] C.E. Rasmussen, C. Williams, Gaussian Processes for Machine Learning, MIT Press, 2006.
[27] D.R. Jones, M. Schonlau, W.J. Welch, Efficient global optimization of expensive black-box functions, J. Global Optim. 13 (1998) 455–492.
[28] D. Lizotte, Practical Bayesian Optimization (Ph.D. thesis), University of Alberta, Edmonton, Alberta, Canada, 2008.
[29] F. Nogueira, Bayesian optimization: Open source constrained global optimization tool for Python, https://github.com/fmfn/BayesianOptimization.
[30] 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, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[31] R. Cavoretto, A. De Rossi, S. Lancellotti, Bayesian approach for radial kernel parameter tuning, J. Comput. Appl. Math. 441 (2024) 115716.
[32] 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, Nat. Methods 17 (2020) 261–272.
[33] D. Lazzaro, L. Montefusco, Radial basis functions for the multivariate interpolation of large scattered data sets, J. Comput. Appl. Math. 140 (2002) 521–536.
[34] R. Renka, R. Brown, Algorithm 792: Accuracy tests of ACM algorithms for interpolation of scattered data in the plane, ACM Trans. Math. Software 25 (1999) 78–94.
[35] 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.
[36] Z.C. Li, H.T. Huang, Y. Wei, Ill-conditioning of the truncation singular value decomposition and the Tikhonov regularization and their application to numerical partial differential equations, Numer. Linear Algebra Appl. 18 (2011) 205–221.
[37] A. Noorizadegan, C.-S. Chen, R. Cavoretto, A. De Rossi, Efficient truncated randomized SVD for mesh-free kernel methods, Comput. Math. Appl. 164 (2024) 12–20.
[38] Y. Wei, P. Xie, L.P. Zhang, Tikhonov regularization and randomized GSVD, SIAM J. Matrix Anal. Appl. 37 (2016) 649–675.
[39] O. Davydov, F. Zeilfelder, Scattered data fitting by direct extension of local polynomials to bivariate splines, Adv. Comput. Math. 21 (2004) 223–271.