A numerical algorithm for multidimensional modeling of scattered data points

A numerical algorithm for multidimensional modelling of scattered data points

Roberto Cavoretto

Abstract In this paper we propose an N-dimensional (Nd) algorithm for surface modelling of multivariate scattered data points. This code is implemented in MATLAB environment to numerically approximate (usually) large data point sets in $\mathbb{R}^N$, for any $N \in \mathbb{N}$. Since we need to organize the points in a Nd space, we build a kd-tree space-partitioning data structure, which is used to efficiently apply a partition of unity interpolant. This global method is combined with local radial basis function approximants and compactly supported weight functions. A detailed design of the partition of unity algorithm and a complexity analysis of the computational procedures are also considered. Finally, in several numerical experiments we show the performances, i.e. accuracy, efficiency and stability, of the Nd interpolation algorithm, considering various sets of Halton data points for $N \leq 5$.

Keywords Surface modelling · Multidimensional algorithms · Partition of unity methods · Multivariate interpolation · Scattered data

Mathematics Subject Classification (2000) 65D05 · 65D15 · 65D17

1 Introduction

In this work we investigate the problem of constructing a numerical partition of unity algorithm which can efficiently be used, at least theoretically, for scattered data interpolation in $\mathbb{R}^N$, for any $N \in \mathbb{N}$. Though the aim is to deal with the case of multidimensional approximation (in general, with $N \geq 3$), we also analyze univariate and bivariate interpolation in order to show that the obtained results are aligned with the ones we find in higher dimensions. Obviously, above all in one dimension we are aware of being in a setting where the use of (local) partition of unity methods is not of great relevance because the number of data points is usually (relatively) small. Moreover, there exist several numerical techniques

R. Cavoretto
Department of Mathematics "G. Peano", University of Torino, via Carlo Alberto 10, I–10123 Torino, Italy
E-mail: roberto.cavoretto@unito.it

involving radial basis functions (RBFs), splines, etc. that under some particular conditions can solve successfully (and sometimes better) an interpolation problem of this type (see e.g. [14,17]).

Hence, researchers often devote their attention to approximate $2d$ or $3d$ surfaces without arriving to design numerical algorithms and analyze their behavior in higher dimensions, e.g. for $N \geq 3$ (see [19,20,23]); here, we face this topic dealing with the general case of the partition of unity interpolation in $\mathbb{R}^N$, for any N. In literature, although much work has recently been done in order to construct fast algorithms using efficient searching procedures based on the partition of the considered domain in strips [2,9], in spherical zones [7,8,10] or in square and cube cells [11,12], these approaches can successfully be applied only for the $2d$ and $3d$ interpolation. In fact, their extentions in highter dimensions turn out to be quite a lot difficult to implement. For this reason, we have turned from that kind of data structures and we have considered the *kd-trees*, which enable us to efficiently organize the points in a N-dimensional (Nd) space. We remark that the kd-tree is a space-partitioning data structure, whose origin goes back to Bentley [4]; see also the references [3,5,16,18,22].

In this paper we propose an Nd algorithm for multivariate approximation of (usually) large sets of scattered data points. This code is designed in MATLAB environment to numerically approximate data points in $\mathbb{R}^N$, for any $N \in \mathbb{N}$. To organize the points in a Nd space, we build a kd-tree data structure, which is used to apply the partition of unity method [23]. This global scheme is combined with local RBF approximants and compactly supported weight functions. A detailed design of this Nd algorithm and a complexity analysis of the computational procedures are here considered in detail. Moreover, we observe that the implemented code is completely automatic and any choice depending on the space dimension has suitably been studied so that this algorithm can work for any N. Finally, numerical results show the performances of the proposed algorithm on various sets of Halton data points for $N \leq 5$.

The paper is organized as follows. In Section 2 we recall the partition of unity scheme which is here combined with local radial basis functions, also focusing on the general problem of interpolation by RBFs. In Section 3, we describe in detail the Nd algorithm for N-variate surface modelling of scattered data points, which is based on a kd-tree space-partitioning data structure. Complexity and stability of the interpolation algorithm are analyzed as well. In Section 4, in order to analyze accuracy, efficiency and stability of the proposed Nd algorithm, we show several numerical experiments, considering various sets of scattered data and different space dimensions. Finally, Section 5 refers to conclusions and future work.

2 Partition of unity interpolation scheme

Let $\mathcal{X}_n = \{\mathbf{x}_i, i = 1, 2, \ldots, n\}$ be a set of distinct data points, arbitrarily distributed in a domain $\Omega \subseteq \mathbb{R}^N$, $N \geq 1$, with an associated set $\mathcal{F}_n = \{f_i, i = 1, 2, \ldots, n\}$ of data values or function values, which are obtained by sampling some (unknown) function $f : \Omega \to \mathbb{R}$ at the data points, i.e., $f_i = f(\mathbf{x}_i)$, $i = 1, 2, \ldots, n$. Thus, we can now give a detailed description of the partition of unity method.

The basic idea of the partition of unity method is to start with a partition of the open and bounded domain $\Omega \subseteq \mathbb{R}^N$ into d subdomains Ω_j such that $\Omega \subseteq \bigcup_{j=1}^{d} \Omega_j$

with some mild overlap among the subdomains. At first, we choose a partition of unity, i.e. a family of compactly supported, non-negative, continuous functions W_j with $\mathrm{supp}(W_j) \subseteq \Omega_j$ such that $\sum_{j=1}^{d} W_j(\mathbf{x}) = 1$, for all $\mathbf{x} \in \Omega$. Then, we consider the global approximant of the form

$$\mathcal{I}(\mathbf{x}) = \sum_{j=1}^{d} R_j(\mathbf{x}) W_j(\mathbf{x}), \qquad \mathbf{x} \in \Omega. \tag{2.1}$$

The local approximant $R_j : \Omega \to \mathbb{R}$ defines a *radial basis function* (RBF) interpolant for each subdomain Ω_j of the form

$$R_j(\mathbf{x}) = \sum_{k=1}^{n} c_k \phi(d(\mathbf{x}, \mathbf{x}_k)) + \sum_{l=1}^{m} d_l p_l(\mathbf{x}), \tag{2.2}$$

where $d(\mathbf{x}, \mathbf{x}_k) = ||\mathbf{x} - \mathbf{x}_k||_2$ is the Euclidean distance, $\phi : [0, \infty) \to \mathbb{R}$ is called *radial basis function*, and $p_1, p_2, \ldots, p_m$ form a basis for the $m = \binom{M-1+N}{M-1}$-dimensional linear space Π_{M-1}^{N} of polynomial of total degree less than or equal to $M-1$ in N variables. Moreover, R_j satisfies the interpolation conditions

$$R_j(\mathbf{x}_i) = f_i, \qquad i = 1, 2, \ldots, n. \tag{2.3}$$

In particular, we observe that if the local approximants satisfy the interpolation conditions at data point $\mathbf{x}_i$, i.e. $R_j(\mathbf{x}_i) = f(\mathbf{x}_i)$, then the global approximant also interpolates at this data point, i.e. $\mathcal{I}(\mathbf{x}_i) = f(\mathbf{x}_i)$, for $i = 1, 2, \ldots, n$. Note that in (2.2) and (2.3) we refer to n data points, even if in practice we should have a smaller number of data points in Ω_j, which we will denote by n_j later.

Solving the j-th interpolation problem (2.3) leads to a system of linear equations of the form

$$\underbrace{\begin{bmatrix} \Phi & P \\ P^T & O \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix}}_{\mathbf{b}}, \tag{2.4}$$

or simply

$$A\mathbf{y} = \mathbf{b},$$

where entries of the interpolation matrix A are

$$\begin{aligned} \Phi_{ik} &= \phi(d(\mathbf{x}_i, \mathbf{x}_k)), \qquad && i, k = 1, 2, \ldots, n, \\ P_{il} &= P_l(\mathbf{x}_i), && i = 1, 2, \ldots, n, \ l = 1, 2, \ldots, m, \end{aligned}$$

$\mathbf{c} = [c_1, c_2, \ldots, c_n]^T$, $\mathbf{d} = [d_1, d_2, \ldots, d_m]^T$, $\mathbf{f} = [f_1, f_2, \ldots, f_n]^T$, $\mathbf{0}$ is a zero vector of length m, O is a $m \times m$ zero matrix. Then, since we require the interpolation problem is well-posed, i.e., a solution to the problem exists and is unique in the interpolation space, we consider the following theorem (see, e.g., [14]).

Theorem 1 *If the function ϕ in* (2.2) *is strictly conditionally positive definite of order M on $\mathbb{R}^N$ and the set $\mathcal{X}_n = \{\boldsymbol{x}_i, i = 1, 2, \ldots, n\}$ of data points form an $(M-1)$-unisolvent one, then the system of linear equation* (2.4) *is uniquely solvable.*

We remark that if $M = 0$ we have strictly conditionally positive definite functions of order zero, i.e. strictly positive definite functions.

Now, we give the following definition (see [23]).

Definition 1 Let $\Omega \subseteq \mathbb{R}^N$ be a bounded set. Let $\{\Omega\}_{j=1}^d$ be an open and bounded covering of Ω. This means that all Ω_j are open and bounded and that Ω is contained in their union. Set $\delta_j = \mathrm{diam}(\Omega_j) = \sup_{\mathbf{x},\mathbf{y}\in\Omega_j} ||\mathbf{x}-\mathbf{y}||_2$. We call a family of nonnegative functions $\{W_j\}_{j=1}^d$ with $W_j \in C^k(\mathbb{R}^N)$ a k-stable partition of unity with respect to the covering $\{\Omega_j\}_{j=1}^d$ if

1) $\mathrm{supp}(W_j) \subseteq \Omega_j$;
2) $\sum_{j=1}^d W_j(\mathbf{x}) \equiv 1$ on Ω;
3) for every $\beta \in \mathbb{N}_0^N$ with $|\beta| \leq k$ there exists a constant $C_\beta > 0$ such that

$$||D^\beta W_j||_{L_\infty(\Omega_j)} \leq C_\beta/\delta_j^{|\beta|},$$

for all $1 \leq j \leq d$.

In order to have an idea of the data point distribution and to understand how uniform are the data sets, we define two common indicators of data regularity: the *separation distance* and the *fill distance*. The former is given by

$$q_{\mathcal{X}_n} = \frac{1}{2}\min_{i\neq k} d(\mathbf{x}_i, \mathbf{x}_k),$$

while the latter, which is a measure of the data distribution, is usually defined as

$$h_{\mathcal{X}_n,\Omega} = \sup_{\mathbf{x}\in\Omega} \min_{\mathbf{x}_k\in\mathcal{X}_n} d(\mathbf{x}, \mathbf{x}_k).$$

In accordance with the statements in [23] we require some additional regularity assumptions on the *covering* $\{\Omega_j\}_{j=1}^d$.

Definition 2 Suppose that $\Omega \subseteq \mathbb{R}^N$ is bounded and $\mathcal{X}_n = \{\mathbf{x}_i, i = 1, 2, \ldots, n\} \subseteq \Omega$ are given. An open and bounded covering $\{\Omega_j\}_{j=1}^d$ is called regular for $(\Omega, \mathcal{X}_n)$ if the following properties are satisfied:

(a) for each $\mathbf{x} \in \Omega$, the number of subdomains Ω_j with $\mathbf{x} \in \Omega_j$ is bounded by a global constant K;
(b) each subdomain Ω_j satisfies an interior cone condition;
(c) the local fill distances $h_{\mathcal{X}_{n_j},\Omega_j}$ are uniformly bounded by the global fill distance $h_{\mathcal{X}_n,\Omega}$, where $\mathcal{X}_{n_j} = \mathcal{X}_n \cap \Omega_j$.

The first property (a) is required to ensure that the sum in (2.1) is actually a sum over at most K summands. Since K is independent of n, unlike d, which should be proportional to n, this is essential to avoid losing convergence orders. Moreover, it is crucial for an efficient evaluation of the global interpolant that only a constant number of local approximants has to be evaluated. Then, it should be possible to locate those K indices in constant time. The second and third properties (b) and (c) are important for employing the estimates on RBF interpolants (see [24]).

Moreover, we are able to formulate the following theorem, which yields the polynomial precision and controls the growth of error estimates (see, e.g., [24]). Here, we denote by $\pi_s^N := \pi_s(\mathbb{R}^N)$ the set of polynomials of degree $s = M - 1$.

Theorem 2 *Suppose that $\Omega \subseteq \mathbb{R}^N$ is compact and satisfies an interior cone condition with angle $\theta \in (0, \pi/2)$ and radius $r > 0$. Let $s \in \mathbb{N}$ be fixed and there exist constants $h_0, C_1, C_2 > 0$ depending only on N, θ, r such that $h_{\mathcal{X}_n,\Omega} \leq h_0$. Then, for all $\mathcal{X}_n = \{\boldsymbol{x}_i, i = 1, 2, \ldots, n\} \subseteq \Omega$ and all $\boldsymbol{x} \in \Omega$, there exist functions $R_k : \Omega \to \mathbb{R}$, $k = 1, 2, \ldots, n$, such that*

(1) $\sum_{k=1}^{n} R_k(\boldsymbol{x})p(\boldsymbol{x}_k) = p(\boldsymbol{x}), \quad$ *for all* $\; p \in \pi_s(\mathbb{R}^N)$*;*
(2) $\sum_{k=1}^{n} |R_k(\boldsymbol{x})| \leq C_1$*;*
(3) $R_k(\boldsymbol{x}) = 0$ *provided that* $||\boldsymbol{x} - \boldsymbol{x}_k||_2 > C_2 h_{\mathcal{X}_n,\Omega}$.

After defining the space $C^k_\nu(\mathbb{R}^N)$ of all functions $f \in C^k$ whose derivatives of order $|\beta| = k$ satisfy $D^\beta f(\mathbf{x}) = \mathcal{O}(||\mathbf{x}||_2^\nu)$ for $||\mathbf{x}||_2 \to 0$, we consider the following convergence result (see, e.g., [14,24]).

Theorem 3 *Let $\Omega \subseteq \mathbb{R}^N$ be open and bounded and suppose that $\mathcal{X}_n = \{\boldsymbol{x}_i, i = 1, 2, \ldots, n\} \subseteq \Omega$. Let $\phi \in C^k_\nu(\mathbb{R}^N)$ be a strictly conditionally positive definite function of order M. Let $\{\Omega_j\}_{j=1}^d$ be a regular covering for $(\Omega, \mathcal{X}_n)$ and let $\{W_j\}_{j=1}^d$ be k-stable for $\{\Omega_j\}_{j=1}^d$. Then the error beetween $f \in \mathcal{N}_\phi(\Omega)$, where $\mathcal{N}_\phi$ is the native space of ϕ, and its partition of unity interpolant (2.1) with $R_j \in \operatorname{span}\{\phi(\cdot) : \boldsymbol{x} \in \mathcal{X}_{n_j}\} + \Pi^N_{M-1}$ can be bounded by*

$$|D^\beta f(\boldsymbol{x}) - D^\beta \mathcal{I}(\boldsymbol{x})| \leq C h_{\mathcal{X}_n,\Omega}^{(k+\nu)/2-|\beta|} |f|_{\mathcal{N}_\phi(\Omega)},$$

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

Note that the partition of unity preserves the local approximation order for the global fit. Hence, we can efficiently compute large RBF interpolants by solving small RBF interpolation problems (in parallel as well) and then combine them together with the global partition of unity $\{W_j\}_{j=1}^d$. This approach enables us to decompose a large problem into many small problems, and at the same time ensures that the accuracy obtained for the local fits is carried over to the global one. In particular, the partition of unity method can be thought as a Shepard's method with higher-order data, since local approximations R_j instead of data values f_j are used.

3 Multidimensional algorithm

In this section we present a new N-variate algorithm for surface modelling of (usually) large sets of scattered data points, which lie in a domain $\Omega \subset \mathbb{R}^N$, for any $N \in \mathbb{N}$. For simplicity, we limit us to consider numerical interpolation on the unit hypercubes $\Omega = [0, 1]^N$, but its extension to hyperrectangles or general domains is obviously possible and straightforward. This Nd code, written in MATLAB environment, is based on a global partition of unity interpolant using local RBF interpolants and compactly supported weight functions. Thus, since we need to organize the points in a Nd space, we build an efficient space-partitioning data structure as the *kd-trees* (see [3,5]). In fact, here we have to be able to answer efficiently two queries, known as *range search* and *containing query*. The computational issues we are considering can briefly be described, as follows:

(i) Given a set of data points $\mathbf{x}_i \in \mathcal{X}_n$ and a subdomain Ω_j, find all points situated in that subdomain, i.e. $x_i \in \mathcal{X}_{n_j} = \mathcal{X}_n \cap \Omega_j$.

(ii) Given $\mathbf{x} \in \Omega$, return all subdomains Ω_j such that $\mathbf{x}_i \in \Omega_j$.

These two items are essential in the design of the partition of unity algorithm, since they require the construction of the kd-trees, which characterize the three phases of the Nd algorithm, i.e. data partition, localization and evaluation stages. Note that the subdomain Ω_j is here a generic region, thus we should think the index j is fixed.

3.1 Data structure: the kd-trees

A kd-tree, short for k-dimensional tree, is a space-partitioning data structure for organizing points in a k-dimensional space. In this work, since we have a N-dimensional space, we should refer to these trees as Nd-trees. However, in order keep the commonly used notation, we will go on to call them kd-trees (see [14]).

The basic idea of the kd-trees is to hierarchically decompose a data point set $\mathcal{X}_n \subseteq \mathbb{R}^N$ in order to obtain a relatively small number of subsets such that each subset contains roughly the same number of data points. In particular, the kd-tree data structure is based on a recursive space subdivision into disjoint rectangular regions, called *boxes*. Moreover, it is characterized by two additional parameters, i.e. the *bucket size* and a *splitting rule*.

Thus, as long as the number of data points associated with an arbitrary node is greater than the bucket size, the box is split into two boxes by an axis-orthogonal hyperplane intersecting this box. In fact, each node in the tree is defined by a splitting hyperplane which turns out to be perpendicular to one of the coordinate axes. Therefore the splitting hyperplanes partition at the median the set of data points into "left" and "right" (or "top"and "bottom") subsets, while the two resulting boxes are the cells associated with the two children of the considered node. This partitioning process is further repeated to the two children using hyperplanes through a different dimension and stops after $\log n$ levels. It follows that each node of the tree is associated with a box and with a subset of data points that are contained in this box, whereas the root node of the tree is associated with the bounding box containing all the set $\mathcal{X}_n$ of data points. If the number of points related to the current box is less than or equal to the bucket size, then the resulting node is a leaf, and these points are stored with the node. We remark that when a point lies on the hyperplane itself it may belong to either child according to the considered splitting rule.

Moreover, we observe that the tree itself is a binary tree with two different types of nodes, i.e. splitting nodes and leaves. The root node is in principle a splitting node (unless the number of points is less than the bucket size), which contains additional and useful information such as the number of data points, the space dimension, and the bucket size. A splitting node stores the integer splitting dimension indicating the coordinate axis orthogonal to the cutting hyperplane. It also stores the splitting value where the hyperplane intersects the axis of the splitting dimension. Moreover, it contains two pointers, one for each child corresponding to the left and right sides of the cutting plane. A leaf node stores the number of points that are associated with this node and an array of the indices of the data points lying in the associated box [24].

Though there are several types of kd-tree splitting rules, here we consider the one whose splitting dimension is given by the dimension of the maximum spread

of the current subset $\mathcal{Y}$ of data points in the current box $\mathcal{B}$. Note that for the root node $\mathcal{Y} \equiv \mathcal{X}_n$ and $\mathcal{B}$ is the bounding box of $\mathcal{X}_n$. The splitting value is the median of the coordinates of $\mathcal{Y}$ along this dimension. Therefore, assuming to have the bounding box of $\mathcal{X}_n$ and the bucket size b, Algorithm 1 describes the procedure of building a kd-tree.

Algorithm 1 Build kd-tree

Input: $\mathcal{X}_n = \{\mathbf{x}_i, i = 1, 2, \ldots, n\}$, set of data points; $\mathcal{B}$, box
Output: ν, root of a kd-tree

```
if (# points of X_n) ≤ b then
    return a leaf storing the indices of these points
else
    Find splitting dimension and value
    Split the current bounding box B into two subboxes B_1 and B_2
            and its points into two subsets of points Y_1 and Y_2
    Apply this algorithm to Y_1 and B_1, resulting in a pointer ν_1
    Apply this algorithm to Y_2 and B_2, resulting in a pointer ν_2
    Create a splitting node ν storing the splitting dimension, the splitting value,
            the left child ν_1 and the right child ν_2
    return ν
end if
```

In Algorithm 1 the most expensive step in each recursive call is to find the splitting value. The median can be obtained in linear time, but it requires to implement rather complicated algorithms. For this reason, we first presort the data set on each coordinate in a preprocessing step, which can be done in $\mathcal{O}(Nn \log n)$ time and needs additional $\mathcal{O}(Nn)$ space at least temporarily to keep the sorted lists. Then we find the median in linear time. Furthermore, we can also construct the two sorted lists for the recursive calls from the given list in linear time. Hence, the resulting time needed to build the tree is $\mathcal{O}(n \log n)$. As a kd-tree is a binary tree with $\mathcal{O}(n)$ leaves and each (internal) node makes use of $\mathcal{O}(1)$ space, the total amount of space is $\mathcal{O}(n)$. Therefore, in order to build the kd-tree for n points in $\mathbb{R}^N$, Algorithm 1 needs $\mathcal{O}(Nn \log n)$ time and $\mathcal{O}(Nn)$ space. As a consequence of this result we have that the kd-tree built in this case has $\mathcal{O}(\log n)$ depth (see [24]).

Let us now consider the range query problem. In Algorithm 2 we describe the pseudocode used in our procedure and suitably adapted to our notation, that is, taking the subdomain Ω_j as a generic region where the index j is fixed. However, the main test in this algorithm is done to check whether Ω_j intersects the box $\mathcal{B}(\nu)$ associated with a node ν. Note that it is not necessary to compute the associated cell $\mathcal{B}(\nu)$ each time. The current cell is maintained through the recursive calls using only the information on the splitting value and the splitting dimension. Therefore, once the kd-tree is build, the range query in Algorithm 2 can be performed in $\mathcal{O}(\log n)$.

Finally, let us analyze the containment query problem. Since kd-trees can be used to build an overlapping domain decomposition, we can efficiently answer the containment query. This can simply be done considering two (instead of one) cutting hyperplanes and assigning all data points on the left of the right cutting plane to the left child and all data points on the right of the left cutting plane to the right child.

Algorithm 2 Range query

Input: $\Omega_j \subseteq \Omega$, query subdomain; ν, root of kd-tree
Output: $\mathcal{X}_{n_j}$, set of points contained in Ω_j

```
if (ν is a leaf) then
   return points of Ω_j stored at ν
else
   ν_1: the left child of ν ← box B_1
   ν_2: the right child of ν ← box B_2
   for i = 1, 2 do
      if (B_i ⊂ Ω_j) then
         return points in the tree rooted at ν_i
      else
         if Ω_j ∩ B_i then
            Apply this algorithm to Ω_j and ν_i
         end if
      end if
   end for
end if
```

3.2 Nd algorithm

Let us now consider in detail the Nd interpolation algorithm.

INPUT: N, space dimension; n, number of data; $\mathcal{X}_n = \{\mathbf{x}_i, i = 1, 2, \ldots, n\}$, set of data points; $\mathcal{F}_n = \{f_i, i = 1, 2, \ldots, n\}$, set of data values.

OUTPUT: $\mathcal{A}_s = \{\mathcal{I}(\tilde{\mathbf{x}}_i), i = 1, 2, \ldots, s\}$, set of approximated values.

`Stage 1.` The set $\mathcal{X}_n$ of data points and the set $\mathcal{F}_n$ of data values are loaded.

`Stage 2.` After computing the number d_{PU} of subdomain points along one direction of Ω by using the rule

$$d_{PU} = \left\lceil \frac{1}{2} \left(\frac{n}{2} \right)^{1/N} \right\rceil, \tag{3.1}$$

and setting

$$d = d_{PU}^N,$$

we construct the set $\mathcal{C}_d = \{\bar{\mathbf{x}}_j, j = 1, 2, \ldots, d\}$ of subdomain points, that is a grid of equally spaced centres of partition of unity subdomains in the unit hypercube. The value in (3.1) depends on both the data point number n and the space dimension N, but it is suitably chosen assuming that the ratio $n/d \approx 2^{N+1}$.

`Stage 3.` The number s_{PU} of evaluation points along one direction is computed by applying the formula in (3.1), taking $s_{PU} = d_{PU}$ and creating another grid of s equally spaced evaluation points, whose set is denoted by $\mathcal{E}_s = \{\tilde{\mathbf{x}}_i, i = 1, 2, \ldots, s\}$.

`Stage 4.` For each subdomain point $\bar{\mathbf{x}}_j$, $j = 1, 2, \ldots, d$, a local spherical subdomain is constructed, whose radius depends on the parameter d_{PU}, i.e.

$$\delta_{subdom} = \frac{\sqrt{2}}{d_{PU}}. \tag{3.2}$$

`Stage 5.` The kd-tree data stuctures are built for the set $\mathcal{X}_n$ of data points and the set $\mathcal{E}_n$ of evaluation points, applying Algorithm 1.

`Stage 6.` For each subdomain Ω_j, $j = 1, 2, \ldots, d$, the range query problem is considered as outlined in Algorithm 2, adopting the related searching procedure which consists of the following two steps:

i) Find all data points (i.e. the set $\mathcal{X}_{n_j}$) belonging to the subdomain Ω_j and construct a local interpolation RBF matrix by $\mathcal{X}_{n_j}$, where n_j denotes the point number of $\mathcal{X}_{n_j}$.
ii) Determine all evaluation points (i.e. the set $\mathcal{E}_{s_j}$) belonging to the subdomain Ω_j and build a local evaluation RBF matrix by $\mathcal{E}_{s_j}$, where s_j is the point number of $\mathcal{E}_{s_j}$.

`Stage 7.` A local RBF interpolant R_j and a weight function W_j, $j = 1, 2, \ldots, d$, is computed for each evaluation point.

`Stage 8.` The global fit (2.1) is applied, accumulating all the R_j and W_j.

In the Nd algorithm the local interpolants are computed by using either globally supported RBFs such as Gaussian and Matérn functions, which are compactly supported on the local subdomain Ω_j, or compactly supported RBFs as the Wendland functions. However, this approach is completely automatic and turns out to be very flexible, since different choices of local approximants (either globally or compactly supported) are allowable.

3.3 Computational cost

The Nd algorithm is based on the construction of kd-tree data structures, which allow us to efficiently find the data points belonging to each subdomain Ω_j, $j = 1, 2, \ldots, d$, so that we can compute local RBF interpolants to be used in the partition of unity scheme. Thus, supposing that the set $\mathcal{X}_n$ of data points is quasi-uniform and the covering $\{\Omega_j\}_{j=1}^d$ is regular and local, that is the size of each subdomain is proportional to $h_{\mathcal{X}_n,\Omega}$, we analyze the complexity of this interpolation algorithm.

At first, from `Stage 1` to `4` we have a sort of preprocessing phase where we automatically load all data sets and define the parameters concerning data, subdomain and evaluation points. In particular, we refer to the rule (3.1) which is obtained under the requirement that the subdomain number d is proportional to the data point number n, taking $n/d \approx 2^{N+1}$. This condition, along with the value of the subdomain radius (3.2), enables the algorithm to efficiently work for any space dimension N.

Then, in `Stage 5` we build the kd-trees applying Algorithm 1, which needs $\mathcal{O}(Nn \log n)$ time and $\mathcal{O}(Nn)$ space for n data points and $\mathcal{O}(Ns \log s)$ time and $\mathcal{O}(Ns)$ space for s evaluation points, whereas in `Stage 6` we make use of the range search process described in Algorithm 2 for each subdomain Ω_j, $j = 1, 2, \ldots, d$, whose running times are $\mathcal{O}(\log n)$ and $\mathcal{O}(\log s)$, respectively. Thus, since the number of centres in each subdomain Ω_j is bounded by a constant (see Definition 2), we need $\mathcal{O}(1)$ space and time for each subdomain to solve the local RBF interpolation problems. In fact, in order to obtain the local RBF interpolants, we have to solve d linear systems of (relatively) small sizes, i.e. $(n_j + m) \times (n_j + m)$, with $\{n_j, m\} << n$, thus requiring a (constant) running time $\mathcal{O}((n_j + m)^3)$,

$j = 1, 2, \ldots, d$, for each subdomain. Besides reporting the points in each subdomain in $\mathcal{O}(1)$, as the number d of subdomains Ω_j is bounded by $\mathcal{O}(n)$, this leads to $\mathcal{O}(n)$ space and time for solving all of them.

Finally, in `Stage 7` and `8` we have to add up a constant number of local RBF interpolants to get the value of the global fit (2.1). This can simply be done in $\mathcal{O}(1)$ time.

3.4 Stability

As to stability, the condition number of the interpolation matrices generated by RBFs depends on both the order of the considered basis functions and the density of the data points. Moreover, the conditioning grows primarily due to the decrease of the separation distance $q_{\mathcal{X}_n}$, and not only necessarily to the increase of the data point number n. Thus, since for most of the subdomains the local separation distance $q_{\mathcal{X}_{n_j}}$ is of the same size (or smallness) as the global separation distance $q_{\mathcal{X}_n}$, the partition of unity method seems to be stable as the global one.

On the other hand, if one keeps fixed the number of data points or, at least, the separation distance, considering instead flatter basis functions with a suitable choice of the shape parameter, then the condition number of the interpolation matrix in (2.4) suffers in almost the same manner. Obviously, a more peaked basis function can be used to improve the condition number in (2.4), but the accuracy of the fit gets worse. In fact, according to the *trade-off principle* we remark that the order of the basis functions should be chosen with great care, because using *standard* bases one cannot have high accuracy and stability at the same time [15]. This order should be enough low when the data density is quite high, because any excessive order has negative effects on stability. For this reason, for low density interpolation data points one can use high-order (e.g., infinitely smooth) basis functions, whereas for high density interpolation data points, to avoid numerical problems, one can use low-order basis functions.

4 Numerical results

In this section we report a few numerical and graphical results coming from experiments obtained by computational procedures developed in MATLAB environment. All the tests have been carried out on a Intel Core 2 Duo Computer (2.1 GHz). In our tests we analize accuracy, efficiency and stability of the Nd algorithm considering some sets of scattered data points, which are contained in the unit hypercube $\Omega = [0, 1]^N$, for $N = 1, 2, \ldots, 5$. They are uniformly random Halton data points generated by using the program `haltonseq.m`, which has been written by D. Dougherty and can be downloaded from the MATLAB Central File Exchange (see [14]). Note that the implemented algorithm is a unique code and, at least theoretically, works well for any N.

Thus, in order to point out accuracy of the Nd algorithm, we compute on each of the considered test functions the Root Mean Square Errors (RMSEs), whose

formula is given by

$$RMSE = \sqrt{\frac{1}{s}\sum_{i=1}^{s} |f(\tilde{\mathbf{x}}_i) - \mathcal{I}(\tilde{\mathbf{x}}_i)|^2}, \tag{4.1}$$

whereas we analyze stability of the considered approach by using the `cond` function of MATLAB, so that one can have a measure of the conditioning of the interpolation matrices generated by RBFs. More precisely, since the local approximation scheme is characterized by the solution of d linear systems of (relatively) small sizes, in order to obtain a good conditioning estimate, we make an average among the conditioning numbers of the d matrices, i.e.

$$Average\ Cond = \frac{1}{d}\sum_{j=1}^{d} cond(A_j), \tag{4.2}$$

where A_j denotes the j-th matrix associated with the subdomain Ω_j.

Moreover, as we will show the results obtained by using different local RBFs, in Table 1 we report a list of some strictly positive definite RBFs with their smoothness degrees. We remark that $\alpha, \epsilon, \delta \in \mathbb{R}^+$ are the *shape parameters*, $r = ||\cdot||_2$ is the Euclidean distance, and $(\cdot)_+$ denotes the truncated power function. We remark that Gaussian C^∞ and Matérn C^4 are globally supported basis functions and strictly positive definite in $\mathbb{R}^N$ for any N, whereas Wendland C^4 and C^2 are compactly supported ones (whose support is $[0, 1/\delta]$) and strictly positive definite in $\mathbb{R}^N$ for $N \leq 3$ (see [24]). Note that the Wendland C^2 function is here just mentioned because it is used as a localizing function of Shepard's weight W_j in the global fit (2.1); conversely, all other RBFs are considered for using them as local interpolants in the partition of unity scheme.

RBF	$\phi(r)$
Gaussian C^∞ (G)	$\mathrm{e}^{-\alpha^2 r^2}, \quad \alpha > 0$
Matérn C^4 (M4)	$\mathrm{e}^{-\epsilon r}(\epsilon^2 r^2 + 3\epsilon r + 3), \quad \epsilon > 0$
Wendland C^4 (W4)	$(1-\delta r)_+^6 \left(35\delta^2 r^2 + 18\delta r + 3\right), \quad \delta > 0$
Wendland C^2 (W2)	$(1-\delta r)_+^4 \left(4\delta r + 1\right), \quad \delta > 0$

Table 1: Strictly positive definite radial basis functions.

4.1 Test case 1

In this subsection we summarize the extensive and detailed investigation we performed to test and verify efficiency, accuracy and stability of the Nd algorithm, showing the numerical results obtained by considering three scattered data sets, which respectively contain $n = 20^N, 40^N, 60^N$, $N = 1, 2, 3$, Halton points.

In the various experiments we analyze the performance of the interpolation algorithm taking the data values by three test functions, known as Franke's functions (see, e.g., [2, 21]), whose analytic expressions are:

$$\begin{aligned} f_1(x_1) &= \frac{3}{4}\mathrm{e}^{-\frac{(9x_1-2)^2}{4}+\frac{25}{16}} + \frac{3}{4}\mathrm{e}^{-\frac{(9x_1+1)^2}{49}-\frac{11}{20}} \\ &+ \frac{1}{2}\mathrm{e}^{-\frac{(9x_1-7)^2}{4}+\frac{9}{16}} - \frac{1}{5}\mathrm{e}^{-(9x_1-4)^2-\frac{25}{4}}, \end{aligned}$$

$$\begin{aligned} f_2(x_1,x_2) &= \frac{3}{4}\mathrm{e}^{-\frac{(9x_1-2)^2+(9x_2-2)^2}{4}} + \frac{3}{4}\mathrm{e}^{-\frac{(9x_1+1)^2}{49}-\frac{9x_2+1}{10}} \\ &+ \frac{1}{2}\mathrm{e}^{-\frac{(9x_1-7)^2+(9x_2-3)^2}{4}} - \frac{1}{5}\mathrm{e}^{-(9x_1-4)^2-(9x_2-7)^2}, \end{aligned}$$

$$\begin{aligned} f_3(x_1,x_2,x_3) &= \frac{3}{4}\mathrm{e}^{-\frac{(9x_1-2)^2+(9x_2-2)^2+(9x_3-2)^2}{4}} + \frac{3}{4}\mathrm{e}^{-\frac{(9x_1+1)^2}{49}-\frac{9x_2+1}{10}-\frac{9x_3+1}{10}} \\ &+ \frac{1}{2}\mathrm{e}^{-\frac{(9x_1-7)^2+(9x_2-3)^2+(9x_3-5)^2}{4}} - \frac{1}{5}\mathrm{e}^{-(9x_1-4)^2-(9x_2-7)^2-(9x_3-5)^2}. \end{aligned}$$

Note that the bivariate and trivariate functions f_2 and f_3 are commonly used in approximation processes to test the behavior of numerical methods and algorithms, whereas the univariate function f_1 has been obtained by f_2 setting $x_2 = 1/2$.

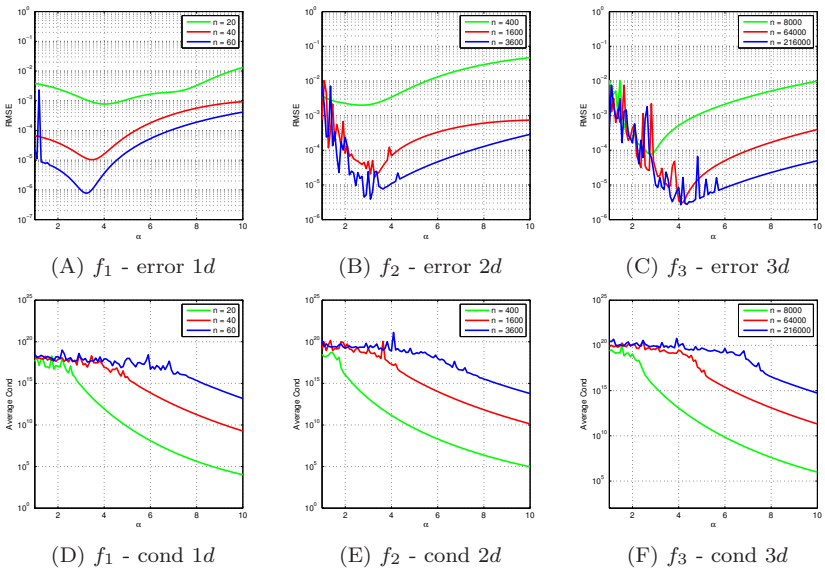
Now, since we are interested in poiting out the efficiency of the proposed interpolation algorithm, in Table 2 we show CPU times (in seconds) obtained by running the Nd algorithm for $N = 1, 2, 3$ as described in Section 3. This table emphasizes the algorithm efficiency characterized by the use of a kd-tree data structure to partition a N-dimensional space. This approach gives a considerable saving of time also when the number of interpolated data points and the space dimension N increase. Furthermore, we remark that the Nd algorithm (with $N = 3$) has also been compared with the trivariate algorithm in [12], turning out in general more efficient than the trivariate one. However, once more we observe that the algorithm in [12] is an "ad hoc" procedure for $3d$ interpolation, whereas the Nd algorithm works well for any N.

N	n	CPU time
	20	0.08
1	40	0.09
	60	0.09
	400	0.18
2	1600	0.34
	3600	0.62
	8000	2.15
3	64000	24.33
	216000	95.93

Table 2: CPU times (in seconds) obtained by running the Nd algorithm.

Then, in order to investigate accuracy and stability, in Figures 1–2 we analyze the behavior of RMSEs and average conditioning computed with the formulas in

(4.1) and (4.2) by varying the shape parameters $\alpha \in [1, 10]$ and $\delta \in [0.1, 4]$ for each of the considered RBFs, i.e. the Gaussian (G) and the Wendland C^4 (W4). We note that in those graphs (left to right) we report the results obtained for the $1d$, $2d$ and $3d$ interpolation, respectively. Furthermore, from this analysis we can observe that the W4-RBF has a greater stability than the G-RBF, but the latter gives us a level of slightly greater accuracy although its interpolation matrices turn out to be subject to ill-conditioning, mainly for small values of α. This behavior is what we expect from theoretical standpoint (see Subsection 3.4), but here it is validated by numerical tests. However, in general, when the G-RBF suffers from ill-conditioning due either to too small values of α or too much data points, the use of the (stable) W4-RBF is strongly preferable and advised. This is further confirmed by making a comparison between the corresponding graphs reported in Figures 1–2. Finally, these graphs allow us to find the optimal values of α and δ, i.e. those values for which we obtain the smallest RMSEs (see Tables 3–5). Note that the average conditioning numbers reported in those tables refer to the optimal values associated with the lowest RMSE.



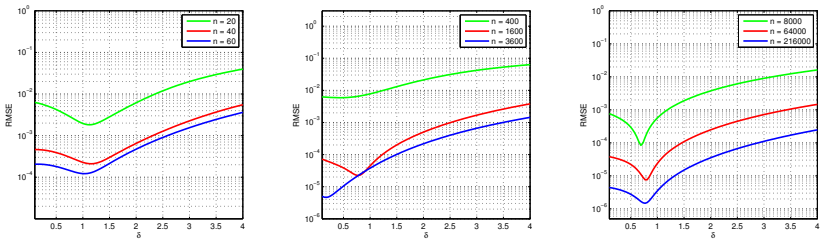
(A) f_1 - error $1d$ (B) f_2 - error $2d$ (C) f_3 - error $3d$

(D) f_1 - cond $1d$ (E) f_2 - cond $2d$ (F) f_3 - cond $3d$

Fig. 1: RMSEs (top) and average conditioning (bottom) obtained by varying the shape parameters for G.

4.2 Test case 2

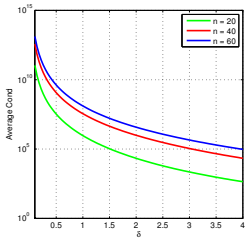
In order to test the performances of the Nd algorithm and understand better its behavior, in these experiments we analyze the results obtained by taking some other data sets $\mathcal{X}_n \subseteq \Omega$ of Halton data points. More precisely, here we consider a



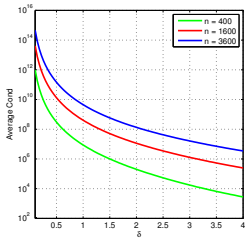
(A) f_1 - error $1d$

(B) f_2 - error $2d$

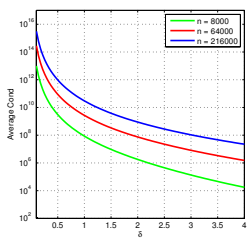
(C) f_3 - error $3d$



(D) f_1 - cond $1d$



(E) f_2 - cond $2d$



(F) f_3 - cond $3d$

Fig. 2: RMSEs (top) and average conditioning (bottom) obtained by varying the shape parameters for W4.

	G			W4		
n	α_{opt}	RMSE	# cond	δ_{opt}	RMSE	# cond
20	4.00	$7.74\text{E}-4$	$8.75\text{E}+11$	1.12	$1.82\text{E}-3$	$5.03\text{E}+05$
40	3.45	$1.04\text{E}-5$	$2.05\text{E}+18$	1.16	$2.10\text{E}-4$	$1.64\text{E}+07$
60	3.27	$7.79\text{E}-7$	$3.69\text{E}+17$	1.05	$1.22\text{E}-4$	$1.03\text{E}+08$

Table 3: RMSEs and average conditioning (# cond) obtained by using optimal values of α and δ for f_1.

	G			W4		
n	α_{opt}	RMSE	# cond	δ_{opt}	RMSE	# cond
400	2.73	$2.00\text{E}-3$	$7.20\text{E}+13$	0.45	$5.91\text{E}-3$	$4.93\text{E}+08$
1600	3.27	$1.68\text{E}-5$	$4.66\text{E}+18$	0.77	$2.24\text{E}-5$	$1.52\text{E}+09$
3600	3.09	$3.88\text{E}-6$	$3.53\text{E}+19$	0.18	$4.64\text{E}-6$	$2.50\text{E}+13$

Table 4: RMSEs and average conditioning (# cond) obtained by using optimal values of α and δ for f_2.

few sets of Halton points whose size n depends on the dimension N, i.e. we take

	G			W4		
n	α_{opt}	RMSE	# cond	δ_{opt}	RMSE	# cond
8000	2.82	7.66E − 5	7.01E + 15	0.69	8.42E − 5	5.87E + 08
64000	4.09	3.09E − 6	2.35E + 19	0.77	7.60E − 6	1.01E + 10
216000	4.09	2.67E − 6	1.24E + 20	0.77	1.48E − 6	1.11E + 11

Table 5: RMSEs and average conditioning (# cond) obtained by using optimal values of α and δ for f_3.

$n = 10^N$, $1 \leq N \leq 5$, focusing on the following N-variate test function [14]

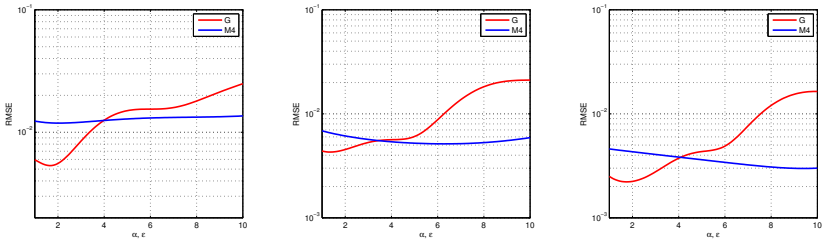
$$g_N(\mathbf{x}) = 4^N \prod_{h=1}^{N} x_h(1 - x_h), \quad \mathbf{x} = (x_1, x_2, \ldots, x_N) \in \Omega.$$

However, to have a more complete view, we have tested other test functions whose results we here omit for shortness, since accuracy and stability have shown similar behaviors.

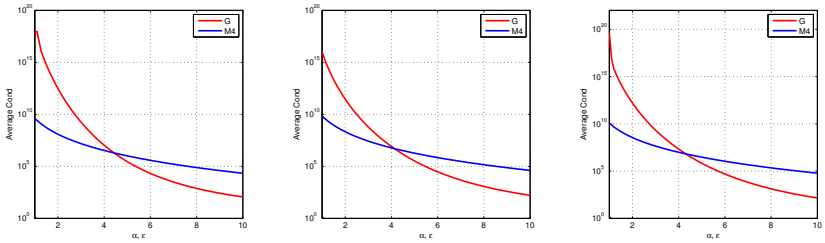
Thus, the purpose of these experiments is firstly to establish how the proposed algorithm works in higher dimensions, i.e. for $N > 3$, but also aiming at further investigating the behavior of RMSEs and average conditioning for lower dimension of space. Therefore, after showing in Table 6 the CPU times (in seconds) computed by running the Nd algorithm for $N \leq 5$, we summarize our numerical study in Figure 3, reporting RMSEs and average conditioning by varying the shape parameters α and ϵ for the Gaussian (G) and the Matérn C^4 (M4), respectively. We remark that in these tests we take the M4 as function of smoothness C^4 instead of the W4, because the latter is only strictly positive definite in $\mathbb{R}^N$ for $N \leq 3$, whereas the M4 function is a strictly positive definite function for any N. However, for shortness, obtaining uniform results in the different space dimensions, we omit the graphs for $1d$ and $2d$ interpolation, only reporting the best values of errors and conditioning numbers (see Table 7). In conclusion, we observe that these experiments confirm the results shown in Subsection 4.1; in fact, though accuracy (errors) can appear worse and stability (conditioning) turns out to be better (above all for the Gaussian case), this depends essentially on the reduction of the number n of data points.

N	n	CPU time
1	10	0.08
2	100	0.14
3	1000	0.28
4	10000	4.48
5	100000	158.99

Table 6: CPU times (in seconds) obtained by running the Nd algorithm.



(A) $n = 1000$ - error $3d$ (B) $n = 10000$ - error $4d$ (C) $n = 100000$ - error $5d$



(D) $n = 1000$ - cond $3d$ (E) $n = 10000$ - cond $4d$ (F) $n = 100000$ - cond $5d$

Fig. 3: RMSEs (top) and average conditioning (bottom) obtained by varying the shape parameters for g_3, g_4 and g_5.

		G			M4		
N	n	α_{opt}	RMSE	# cond	ϵ_{opt}	RMSE	# cond
1	10	1.00	$1.09\text{E}-2$	$3.01\text{E}+16$	1.00	$2.15\text{E}-2$	$9.06\text{E}+07$
2	100	1.00	$9.27\text{E}-3$	$1.37\text{E}+16$	6.90	$2.70\text{E}-2$	$4.02\text{E}+04$
3	1000	1.64	$5.34\text{E}-3$	$1.20\text{E}+14$	2.09	$1.19\text{E}-2$	$9.85\text{E}+07$
4	10000	1.36	$4.29\text{E}-3$	$9.12\text{E}+13$	6.27	$5.14\text{E}-3$	$5.50\text{E}+05$
5	100000	1.73	$2.22\text{E}-3$	$1.65\text{E}+13$	9.45	$2.98\text{E}-3$	$8.43\text{E}+04$

Table 7: RMSEs and average conditioning (# cond) obtained by using optimal values of α and ϵ for g_N.

5 Conclusions and future work

In this paper we present a new partition of unity algorithm for surface approximation of scattered data sets in $\mathbb{R}^N$, for any N. This Nd algorithm is based on the combination of local RBF interpolants with compactly supported Shepard's weight functions. In particular, the optimized implementation of this code and the use of kd-tree data structures allow us to efficiently organize the points in

any Nd space. In fact, this algorithm, which is completely automatic and at least theoretically works well for any N, is designed in such a way that one can model multidimensional surfaces as it is often required in applications. Complexity analysis and numerical experiments show good performances of the Nd algorithm, which is tested on some sets of Halton data points for $N \leq 5$.

As future work we are going to propose an adaptive Nd algorithm based on partition of unity interpolants. This approach should allow us to suitably create subdomains of variable size so that one can efficiently interpolate irregularly distributed data points. This means that we can locally increase (or decrease) the radius of subdomains according to the data distribution, whatever it is. Finally, we could also think to generalize this algorithm for modelling discontinuous surfaces as in [1] or consider different approaches as shown in [6,13].

Acknowledgements The author thanks the support of the Department of Mathematics "G. Peano", University of Torino, project "Numerical analysis for life sciences" (2012).

References

1. Allasia, G., Besenghi, R., Cavoretto, R.: Adaptive detection and approximation of unknown surface discontinuities from scattered data. Simulat. Modell. Pract. Theory **17**, 1059–1070 (2009)
2. Allasia, G., Besenghi, R., Cavoretto, R., De Rossi, A.: Scattered and track data interpolation using an efficient strip searching procedure. Appl. Math. Comput. **217**, 5949–5966 (2011)
3. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J. ACM **45**, 891–923 (1998)
4. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Comm. ACM **18**, 509–517 (1975)
5. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry. Berlin, Springer (1997)
6. Caira, R., Dell'Accio, F., Di Tommaso, F.: On the bivariate Shepard-Lidstone operators. J. Comput. Appl. Math. **236**, 1691–1707 (2012)
7. Cavoretto, R., De Rossi, A.: Fast and accurate interpolation of large scattered data sets on the sphere. J. Comput. Appl. Math. **234**, 1505–1521 (2010)
8. Cavoretto, R., De Rossi, A.: Spherical interpolation using the partition of unity method: An efficient and flexible algorithm. Appl. Math. Lett. **25**, 1251–1256 (2012)
9. Cavoretto, R.: A unified version of efficient partition of unity algorithms for meshless interpolation. In: Simos, T.E., et al. (eds.), Proceedings of the ICNAAM 2012, AIP Conf. Proc., vol. 1479, Melville, New York, pp. 1054–1057 (2012)
10. Cavoretto, R., De Rossi, A.: Achieving accuracy and efficiency in spherical modelling of real data. Math. Methods Appl. Sci. (2013), in press. DOI: 10.1002/mma.2906
11. R. Cavoretto, A. De Rossi, A meshless interpolation algorithm using a cell-based searching procedure. Submitted for publication (2013)
12. Cavoretto, R., De Rossi, A.: A trivariate interpolation algorithm using a cube-partition searching procedure. Submitted for publication (2013)
13. Costabile, F.A., Dell'Accio, F., Di Tommaso, F.: Complementary Lidstone interpolation on scattered data sets. Numer. Algorithms **64**, 157–180 (2013)
14. Fasshauer, G.E.: Meshfree Approximation Methods with MATLAB. World Scientific Publishers, Singapore (2007)
15. Fasshauer, G.E.: Positive definite kernels: past, present and future. Dolomites Res. Notes Approx. **4**, 21–63 (2011)
16. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Software **3**, 209–226 (1977)
17. Iske, A.: Scattered data approximation by positive definite kernel functions. Rend. Sem. Mat. Univ. Pol. Torino **69**, 217–246 (2011)

18. Mount, D.M.: ANN Programming Manual. Maryland, College Park (1998)
19. Nguyen, V.P., Rabczuk, T., Bordas, S., Duflot, M.: Meshless methods: A review and computer implementation aspects. Math. Comput. Simulation **79**, 763–813 (2008)
20. Pouderoux, J., Tobor, I., Gonzato, J.-C., Guitton, P.: Adaptive hierarchical RBF interpolation for creating smooth digital elevation models. In: Pfoser, D., et al. (eds.), GIS 2004: Proceedings of the Twelfth ACM International Symposium on Advances in Geographic Information Systems, pp. 232–240 (2004)
21. Renka, R.J.: Multivariate interpolation of large sets of scattered data. ACM Trans. Math. Software **14**, 139–148 (1988)
22. Samet, H.: The Design and Analysis of Spatial Data Structures. Reading, Addison-Wesley (1990)
23. Wendland, H.: Fast evaluation of radial basis functions: Methods based on partition of unity. In: Chui, C.K., et al. (eds.), Approximation Theory X: Wavelets, Splines, and Applications, Vanderbilt Univ. Press, Nashville, TN, pp. 473–483 (2002)
24. Wendland, H.: Scattered Data Approximation. Cambridge Monogr. Appl. Comput. Math., vol. 17, Cambridge Univ. Press, Cambridge (2005)