AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Fast computation of triangular Shepard interpolants

This is a pre print version of the following article:

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1685142 since 2019-07-28T09:45:45Z

Published version:

DOI:10.1016/j.cam.2018.03.012

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

17 April 2024

This is the author's final version of the contribution published as:

[Roberto Cavoretto, Alessandra De Rossi, Francesco Dell'Accio, Filomena Di Tommaso, Fast computation of triangular Shepard interpolants, Journal of Computational and Applied Mathematics, 354, 2019, pagg. 457-470, https://doi.org/10.1016/j.cam.2018.03.012]

The publisher's version is available at:

[https://www.sciencedirect.com/science/article/pii/S0377042718301584?via%3Dihub]

When citing, please refer to the published version.

Link to this full text:

[http://hdl.handle.net/2318/1685142]

This full text was downloaded from iris-Aperto: https://iris.unito.it/

iris-AperTO

University of Turin's Institutional Research Information System and Open Access Institutional Repository

Fast computation of triangular Shepard interpolants

Roberto Cavoretto[a], Alessandra De Rossi[a], Francesco Dell'Accio[b], Filomena Di Tommaso[b]

[a]*Department of Mathematics "Giuseppe Peano", University of Torino, via Carlo Alberto 10, 10123 Torino, Italy*
[b]*Department of Mathematics and Computer Science, University of Calabria, via P. Bucci cubo 30 A, 87036 Rende (CS), Italy*

Abstract

In this paper we present an efficient algorithm for the computation of triangular Shepard interpolation method. More precisely, it is well known that the triangular Shepard method reaches an approximation order better than the Shepard one [1], but it needs to identify useful general triangulation of the node set. Here we propose a searching technique used to detect and select the nearest neighbor points in the interpolation scheme [2, 3]. It consists in determining the closest points belonging to the different neighborhoods and subsequently applies to the triangulation-based approach. Numerical experiments and some geological applications show efficiency and accuracy of the interpolation procedure.

Keywords: scattered data interpolation, triangular Shepard method, fast computation, approximation algorithms
2010 MSC: 65D05, 65D15, 41A05

1. Introduction

The Shepard method [4] is a well known interpolation scheme to approximate a target function $f : \mathbb{R}^2 \to \mathbb{R}$, when a finite set of its values $F_n = \{f_j = f(x_j), j = 1, \ldots, n\}$ is known on a set $X_n = \{x_1, \ldots, x_n\}$ of data sites or nodes which do not obey any regularity between their relative locations. A set of such data is called *scattered*. Such kind of situations occurs when dealing with real world problems, as for example the reconstruction of geological surfaces and many other ones, see for example [5, 6, 7, 8] and the references therein. The Shepard method consists in a linear combination of the functional values f_j, with non negative coefficients (or weight functions or basis functions) which are inverse distances to the scattered points and form a partition of unity. It is very easy to implement and does not require the solution of any linear system, as opposed to what happens with other well known scattered data interpolation schemes, such as radial basis functions [9, 5, 6, 10]. As a consequence, it is very suited in the interpolation of large sets of scattered data due to its computational efficiency. However, a drawback of the Shepard method is its low polynomial precision (only constant) that badly affects the reconstructed surface. To overcome this drawback several variants of the Shepard method have been introduced [11, 12, 13, 14, 15, 8, 16] by combining Shepard basis functions with local polynomial interpolants with higher polynomial reproduction properties, even by using supplementary derivative data, if given. The triangular Shepard method [17] combines triangle-based weight functions with local linear interpolants and reaches linear precision without using derivative data. Similarly to the Shepard basis functions, the triangle-based basis functions are the product of the inverse distances from the vertices of triangles and form a partition of unity. However, for achieving a good accuracy of approximation, it needs to identify useful general triangulation of the node set [1].

A crucial point in any local interpolation scheme is to have an efficient organization of the scattered data. To this aim, in literature, techniques known as kd-trees, which are not implemented for a specific

Email addresses: `roberto.cavoretto@unito.it` (Roberto Cavoretto), `alessandra.derossi@unito.it` (Alessandra De Rossi), `francesco.dellaccio@unical.it` (Francesco Dell'Accio), `ditommaso@mat.unical.it` (Filomena Di Tommaso)

interpolation method, have already been designed [18, 5]. In this paper, we propose the use of a versatile partitioning structure, called the block-based partitioning structure, and the related searching procedure, given in [2], which are suitably adapted to triangular Shepard interpolation. Such routine enables to deal with a very large number of data, with a low computational complexity. Our procedure consists in partitioning the domain with several non overlapping small square blocks. By recursive calls to a sorting routine, the scattered data points are thus distributed among the different blocks. After storing the data in such blocks, an optimized searching technique is applied to select the nearest neighbor points, thus enabling us to find a suitable triangulation.

The paper is organized as follows. In Section 2 we consider interpolation using the triangular Shepard method and the selection of the compact triangulation. Section 3 is devoted to present the searching technique used to detect and select the nearest neighbor points in our interpolation scheme. In Section 4 the interpolation algorithm and the analysis of the computational cost are presented. Section 5 shows some numerical results obtained by using test functions. In Section 6 we provide a few numerical experiments involving geological applications. Finally, Section 7 deals with conclusions and future work.

2. Interpolation on compact triangulations

2.1. Triangular Shepard method

A variant of the Shepard method, called the triangular Shepard method [17], was introduced by Little in 1983 as a convex combination of the linear interpolants on a set of triangles. More in detail, let $X_n = \{x_1, \dots, x_n\}$ be a set of nodes or data points of $\mathbb{R}^2$ with associated the set $F_n = \{f_1, \dots, f_n\}$ of function or data values, and let $T_m = \{t_1, \dots, t_m\}$ be a triangulation of the node set X_n. With the term *triangulation* here we mean a set of triangles $\{t_j\}_{j=1,\dots,m}$ with vertices $\{x_{j_1}, x_{j_2}, x_{j_3}\}$ which are nodes of X_n such that each point of X_n is the vertex of at least a triangle. Little did not give any indication on the choice of the triangulation T_m. On the other hand, as later specified [1], the triangulation can satisfy particular conditions, for instance it can form a *Delaunay triangulation* of X_n or, more in general, can constitute a so called *compact triangulation*, that is the triangles may overlap or be disjoint. The last condition, being less restrictive than the first one, implies that the triangulation T_m is composed by a significant smaller number of triangles and allows the method to be meshless. The triangular Shepard method is defined by

$$K_\mu[f](x) = \sum_{j=1}^{m} B_{\mu,j}(x) L_j[f](x), \quad \mu > 0, \tag{1}$$

where

$$L_j[f](x) = f_{j_1}\lambda_{j,j_1} + f_{j_2}\lambda_{j,j_2} + f_{j_3}\lambda_{j,j_3}$$

is the linear interpolant on the vertices of t_j, $j = 1, \dots, m$, expressed in barycentric coordinate related to the triangle t_j and the weight functions $B_{\mu,j}(x)$ are defined by

$$B_{\mu,j}(x) = \frac{\displaystyle\prod_{\ell=1}^{3} \frac{1}{\|x - x_{j_\ell}\|^\mu}}{\displaystyle\sum_{k=1}^{m}\prod_{\ell=1}^{3} \frac{1}{\|x - x_{k_\ell}\|^\mu}}, \qquad j = 1, \dots, m. \tag{2}$$

The triangular Shepard operator (1) exceeds the Shepard method both in polynomial precision and esthetic behavior. In fact, as the Shepard method it interpolates on all data (x_j, f_j) since

$$B_{\mu,j}(x_i) = 0, \text{ for each } x_i \notin \{x_{j_1}, x_{j_2}, x_{j_3}\},$$

and

$$\sum_{j \in J_i} B_{\mu,j}(x_i) = 1,$$

where $J_i = \big\{k \in \{1,\dots,m\} : i \in \{k_1,k_2,k_3\}\big\}$ is the set of indices of all triangles which have x_i as a vertex. In addition, while the Shepard method reproduces only constant functions, the triangular Shepard method reproduces polynomials up to the degree 1, since the polynomial operator $L_j[\cdot](x)$ satisfies this property and the basis functions (2) are non negative and form a partition of unity

$$\sum_{j=1}^{m} B_{\mu,j}(x) = 1.$$

The better polynomial precision of the triangular Shepard method reflects on a higher order of approximation. As shown in [1] the triangular Shepard method reaches quadratic approximation order (for $\mu > 4/3$) while the Shepard method achieves at most linear approximation order [19].

2.2. Selection of the compact triangulation

The theoretical study of the approximation order given in [1] shows that the accuracy of approximation of the operator $K_\mu[f](x)$ depends on the distribution of nodes and on the distribution and form of triangles. More in detail, we denote by Ω a compact convex domain which contains X_n and by $C^{1,1}(\Omega)$ the class of differentiable functions $f\colon \Omega \to \mathbb{R}$ whose partial derivatives are Lipschitz-continuous of order 1, equipped with the seminorm

$$\|f\|_{1,1} = \sup\left\{ \frac{\left|\frac{\partial f}{\partial x^{1-\alpha}\partial y^{\alpha}}(u) - \frac{\partial f}{\partial x^{1-\alpha}\partial y^{\alpha}}(v)\right|}{\|u-v\|} : u,v \in \Omega, u \neq v, \alpha \in \{0,1\} \right\}.$$

Moreover, we denote by

$$R_r(y) = \{x \in \mathbb{R}^2 : \|x-y\|_\infty \leq r\}$$

the axis-aligned closed square with centre y and edge length $2r$ and by $V(t)$ the set of vertices of a triangle $t \in T_m$. A small value of

$$h' = \inf\{r > 0 : \forall x \in \Omega \ \exists t \in T_m : R_r(x) \cap V(t) \neq \emptyset\}$$

corresponds to a rather uniform triangles distribution. However, it does not exclude the presence of large triangles, which cannot occur if the value

$$h'' = \inf\{r > 0 : \forall t \in T_m \ \exists x \in \Omega : t \subset R_r(x)\},$$

is also small. In fact, each triangle is contained in a square with edge length $2h$, where

$$h = \max\{h', h''\}.$$

The bound for the approximation error

$$\|f(x) - K_\mu[f](x)\| \leq CM\|f\|_{1,1}h^2, \qquad x \in \Omega,$$

involves other two constants, namely M and C. The constant M is the maximum number of triangles with at least one vertex in some square with edge length $2h$. If there are not clusters of triangles, M is rather small. The constant C is related to the parameter μ and both to the number and the shape of triangles. By denoting with h_j and A_j the maximum edge length and twice the area of the triangle t_j, respectively, we get

$$C = \sqrt{3}^{3m\mu} \max_{j=1,\dots,m} \left\{ \frac{h_j^2}{A_j} \right\}.$$

Useful compact triangulations of the node set X_n are identified by taking into account previously specified remarks. Clearly, we can act on h' and on M, eventually by discarding some nodes of the given node set X_n, and we can avoid duplicate triangles making m about one-third the corresponding value in the case of Delaunay triangulation. Moreover, the control of the shape of the triangles allows to reduce the approximation error. In Figure 1 we display a set of uniformly random 80 Halton points [20] in the square domain $R = [0,1] \times [0,1]$ and a compact triangulation with vertices the given points. In Figure 2 we show the plot of a basis function for a non overlapping triangle (top) and for an overlapping triangle (bottom).

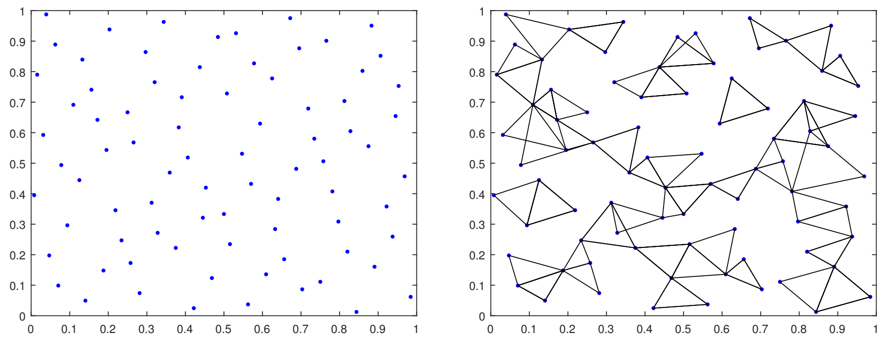


Figure 1: Uniformly random 80 Halton points in $R = [0, 1] \times [0, 1]$ (left) and a compact triangulation with vertices the given points (right).

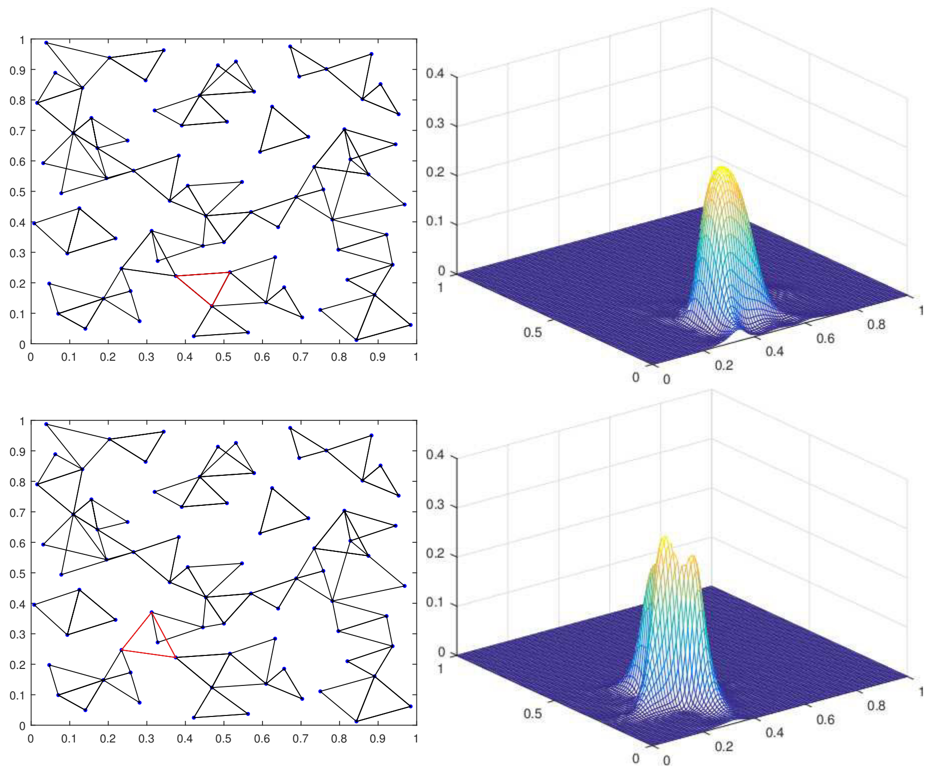


Figure 2: The basis function $B_{\mu,j}$ for a non overlapping triangle (top) and for an overlapping triangle (bottom).

3. Localizing, partitioning and searching techniques

This section is devoted to the presentation of the searching technique used to detect and select the nearest neighbor points in our 2D interpolation scheme [2] (for the 3D scheme see also [21]). This procedure has already been applied in [3] and further extended in [22]. Such a technique is based on the construction of a block-based partitioning structure, which allows to efficiently find all the points belonging to a given neighborhood. Although the searching procedure and the related structure can be applied to generic bivariate domains, for our purpose we describe this localizing technique focusing on the unit square, i.e. on a domain $R = [0, 1] \times [0, 1] \subseteq \mathbb{R}^2$.

3.1. Localizing technique

First, we define a circular neighborhood of radius

$$\delta = \frac{2}{d},$$

with

$$d = \left\lfloor \frac{\sqrt{n}}{2} \right\rfloor, \tag{3}$$

where each neighborhood is centred at a data point belonging to R. Note that the larger (smaller) the value of d is, the finer (coarser) the structure becomes. The value (3) is set extending the definition given in [2].

3.2. Partitioning structure

In order to determine the closest points belonging to the different neighborhoods and consequently apply our triangulation-based approach, we propose a new structure that partitions the domain in square blocks. Such technique results in an effective searching procedure which is quite efficient from a computational viewpoint. For this scope we partition the domain R with b^2 square blocks, b being the number of blocks along one side of the unit square defined as

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

It follows that the side of each square block turns out to be equal to the neighborhood radius. Hence, such a choice (seemingly trivial) allows us to examine in the searching process only a small number of blocks, significantly reducing the computational effort compared to standard or more advanced searching procedures such kd-trees [18, 10]. In fact, our searching routine is performed in a constant time, independently from the initial number of nodes considered. Moreover, in the partitioning technique we number the blocks of square shape from 1 to b^2, following the lexicographic order “bottom to top, left to right”. By repeatedly using a quicksort routine, we can thus split by the block-based partitioning structure the set X_n in the b^2 square blocks, as to easily be able to construct the subsets X_{n_k}, $k = 1, \ldots, b^2$, where X_{n_k} contains the points belonging to nine blocks: the k-th block and its eight neighboring blocks, see Figure 3 (top). In such framework, we are able to get an optimal procedure to find the interpolation nodes closest to each of points.

3.3. Searching procedure

After organizing the nodes in square blocks, in order to compute local fits, i.e. interpolants on each neighborhood, we need to answer the following queries, known respectively as *containing query* and *range search* problems:

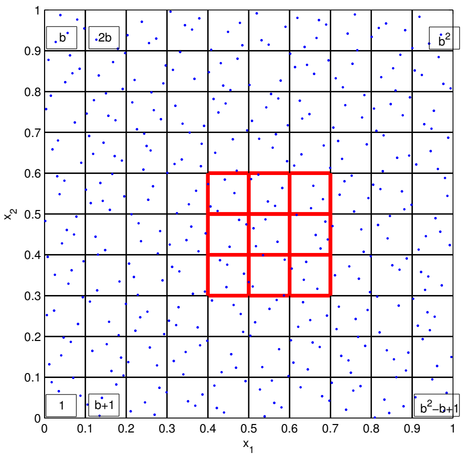
- given a node belonging to R, return the k-th square block containing the node;
- given a set of nodes that belong to X_n and a neighborhood, find all points located in that neighborhood.

In so doing, we remark that square blocks are generated by the intersection of two families of orthogonal strips. The former numbered from 1 to b are parallel to the x_2-axis, while the latter again numbered from 1 to b are parallel to the x_1-axis. Note that from these two families of crossed strips we get the partitioning structure described in the previous subsection.

Given a node in the neighborhood, the block-based containing query provides the index of the block containing such point. Therefore, fixed a neighborhood node, if k_m is the index of the strip parallel to the subspace of dimension one generated by x_p, $p = 1, 2$ and $p \neq m$, containing the m-th coordinate of the neighborhood node, then the index of the k-th block containing the point is given by

$$k = (k_1 - 1)\, b + k_2.$$

As an example, the neighborhood node plotted in Figure 3 (bottom) belongs to the k-th block, with $k = 5b + 5$; in fact, here $k_1 = 6$, $k_2 = 5$ and $b = 10$.



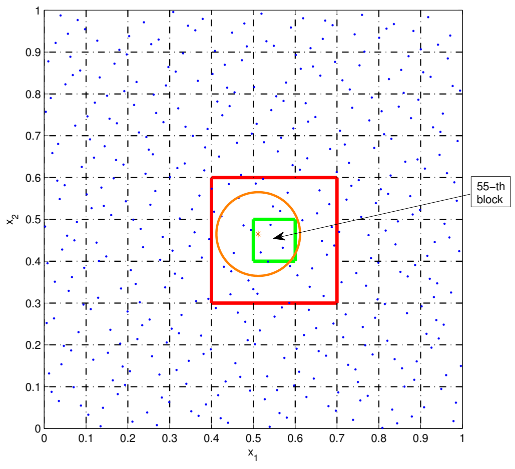


Figure 3: Example of 2D block-based partitioning structure with nodes (blue): on top the partitioning structure with the k-th block and its eight neighboring blocks (red); on bottom the k-th block (green, with $k = 55$) and a neighborhood associated with a node denoted by $\star$ (orange) belonging to the k-th block.

After answering the first query, for each neighborhood given the searching routine allows to determine all nodes belonging to the neighborhoods and, among them, we take a fixed number of points consisting of the n_w nearest neighbors. Specifically, supposing that the node neighborhood belongs to the k-th block, the block-based searching procedure searches for all data lying in the k-th block and in its eight neighboring blocks, see Figure 3 (bottom). Additionally, when a block lies on the boundary of the domain R, in the searching process the partitioning structure based on square blocks enables to further reduce the number of neighboring blocks to be examined.

4. Interpolation algorithm and computational complexity

In this section we describe in a pseudo-code the interpolation algorithm (see Table 1), also analyzing the computational complexity of our partitioning and searching procedures. So we first consider complexity required to build the partitioning structure, and then we focus on the searching procedure.

The localization phase presented in Subsection 3.1 is a sort of data pre-processing which is not involved in complexity cost. So we address our attention on the partitioning structure used to organize the n nodes

```
INPUTS: $n$, number of data; $X_n = \{x_i, i = 1, \dots, n\}$, set of data points; $F_n = \{f_i, i = 1, \dots, n\}$, set of data values; $n_e$, number of evaluation points; $n_w$, localizing parameter.
OUTPUTS: $A_{n_e} = \{K_\mu[f](z_i), i = 1, \dots, n_e\}$, set of approximated values.
Step 1: Generate a set $Z_{n_e} = \{z_1, \dots, z_{n_e}\} \subseteq R$ of evaluation points.
Step 2: For each point $x_i$, $i = 1, \dots, n$, construct a neighborhood of radius
    $$\delta = \frac{2}{d}, \quad \text{with} \quad d = \left\lfloor \frac{\sqrt{n}}{2} \right\rfloor. \qquad (4)$$
Step 3: Compute the number $b$ of blocks defined as
    $$b = \left\lceil \frac{1}{\delta} \right\rceil.$$
Step 4: Build the block-based data structure for the set $X_n$ of data points.
Step 5: For each neighborhood or node (i.e., the neighborhood centre), solve the containing query and the range search problems to find all data points $X_{n_k}$, $k = 1, \dots, b^2$, belonging to the nine nearby blocks.
Step 6: For each node $x_i \in X_n$, fix its $n_w$ nearest neighbors $\mathcal{N}(x_i) \subset X_n$. Among the
    $$\frac{n_w\,(n_w - 1)}{2}$$
    triangles with a vertex in $x_i$ and other two vertices in $\mathcal{N}(x_i)$, choose the one which locally reduces the bound for the error of the local linear interpolant
    $$2\|x - x_{j_1}\|^2 + 4h_j \frac{h_j^2}{A_j} \|x - x_{j_1}\|.$$
Step 7: Compute the local basis function $B_{\mu,j}(z)$, $j = 1, \dots, m$, at each evaluation point $z \in Z_{n_e}$.
Step 8: Compute the linear interpolants $L_j[f](z)$, $j = 1, \dots, m$, at each evaluation point $z \in Z_{n_e}$.
Step 9: Apply the triangular Shepard method (1) and evaluate the surface at the evaluation points $z \in Z_{n_e}$.
```

Table 1: Algorithm performing the triangular Shepard method (1) using the block-based partitioning structure and the related searching procedure.

in blocks. We remark that in the assessment of the total computational cost we should also consider the cost associated with the storing of the evaluation points, even if they are not involved in the partitioning and searching phases. For this reason, here we essentially refer to the interpolation nodes.

As described in Subsection 3.2, the partitioning structure makes use of a quicksort routine which requires $\mathcal{O}(n \log n)$ time complexity and $\mathcal{O}(\log n)$ space, where n is the number of elements to be sorted. Specifically the block data structure is based on recursive calls to the MATLAB `sortrows.m` routine. Hence, in order to analyze the complexity of our procedure, we denote by $\tilde{n}_k$ the number of nodes belonging to the k-th strip. So the computational cost is given by

$$\mathcal{O}\left(n \log n + \sum_{k=1}^{b} \tilde{n}_k \log \tilde{n}_k \right). \qquad (5)$$

Denoting by n/b the average number of points lying in the b strips, from (5) we can write

$$\mathcal{O}\left(n\log n + n\log\frac{n}{b}\right) \approx \mathcal{O}\left(n\log n + n\log(n\delta)\right).$$

From definition of the neighborhood radius in (4), we get the estimate

$$\mathcal{O}\left(\frac{3}{2}n\log n + 2n\right).$$

Additionally, in the block-based partitioning scheme a sorting routine on indices is used to order them. Such procedure is applied with an optimized routine for integers requiring $\mathcal{O}(n)$ time complexity.

To analyze the complexity of the searching procedure, we denote by n_k the number of points belonging to the nine neighboring blocks. Then, since for each neighborhood a quicksort procedure is used to order distances, the routine requires $\mathcal{O}(n_k \log n_k)$ time complexity. Observing that the nodes in nine blocks are about $9n/b^2$ and considering the definitions of b and δ, the complexity can be estimated by

$$\mathcal{O}\left(\frac{9n}{b^2}\log\frac{9n}{b^2}\right) \approx \mathcal{O}(1).$$

The latter estimate follows from the fact that we built a partitioning structure strictly related to the size of the neighborhood. For this reason, the number of points is about constant, independently from the initial value n. Thus using a number of blocks depending both on the number and the size of such neighborhoods, the searching procedure involves a constant number of points, i.e. those belonging to nine blocks.

5. Numerical experiments

In this section we report the performance of our interpolation algorithm which is measured through numerical experiments. All these tests have been carried out on a laptop with an Intel(R) Core i7 5500U CPU 2.40GHz processor and 8.00GB RAM.

In the following we focus on a wide series of experiments, which consist in solving very large interpolation problems via the triangular Shepard interpolant (1). This analysis is thus carried out considering two different distributions of scattered data points contained in the unit square $R = [0,1]^2 \subset \mathbb{R}^2$, with the number n of interpolation points varying from 10 000 to 80 000. Specifically, as interpolation nodes we take some sets of uniformly random Halton points generated through the Matlab program `haltonseq.m` [5], and pseudo-random points obtained by using the `rand` Matlab command, see Figure 4, left to right. Moreover, the computation of interpolation errors is done on a grid of $n_e = 51 \times 51$ evaluation points and using as localizing parameter $n_w = 10$.

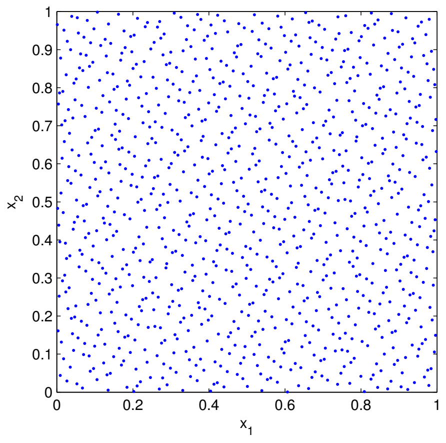
In the various tests we show the performance of our interpolation scheme taking the data values by two bivariate test functions [23]. The former is the well known Franke's function

$$\begin{aligned} f_1(x_1,x_2) = {} & 0.75\exp\left[-\frac{(9x_1-2)^2+(9x_2-2)^2}{4}\right] + 0.50\exp\left[-\frac{(9x_1-7)^2+(9x_2-3)^2}{4}\right] \\ & + 0.75\exp\left[-\frac{(9x_1+1)^2}{49} - \frac{(9x_2+1)}{4}\right] - 0.20\exp\left[-(9x_1-4)^2-(9x_2-7)^2\right], \end{aligned}$$

while the latter is given by

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

Notice that these functions are commonly used in approximation processes to test and validate new methods and algorithms, then making them usable in the field of applications.



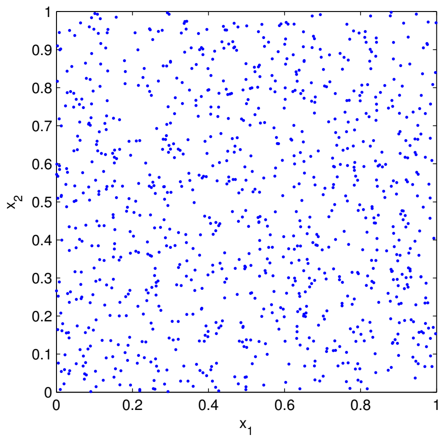


Figure 4: Example of uniformly random Halton points (left) and pseudo-random MATLAB points (right).

In order to investigate the accuracy of our method, we compute the Maximum Absolute Error (MAE) and the Root Mean Square Error (RMSE), whose formulas are

$$\text{MAE} = \max_{1 \le i \le n_e} e_i, \qquad \text{RMSE} = \sqrt{\frac{1}{n_e}\sum_{i=1}^{n_e} {e_i}^2},$$

where

$$e_i = |f(z_i) - K_2[f](z_i)|, \tag{6}$$

$z_i \in Z_{n_e}$ being an evaluation point in R.

In Tables 2–5 we show MAEs and RMSEs that decrease as the number n of nodes increases. However, comparing the errors obtained with the two data distributions, we observe a better accuracy achieved when we use Halton points. This is essentially due to greater uniformity of Halton nodes than pseudo-random MATLAB points, as Figure 4 highlights. Analyzing the error behavior with the test functions f_1 and f_2, we stress similar results in terms of precision of the interpolation scheme, even if the oscillatory nature of function f_2 leads to a slightly lessen accuracy. As for, instead, efficiency of our algorithm we report the CPU times computed in seconds, comparing the performance of our fast procedure based on the block-based searching technique (t_{new}) with a standard implementation where one computes all the distances between the scattered points (t_{old}). From this study we point out the great improvement in terms of computational efficiency when the block-based partitioning and searching procedures are used. Moreover, we remark that the computation of errors and times though the standard procedure is not allowed by MATLAB for $n > 20\,000$ points, because it is very expensive from a computational viewpoint and memory required is not enough. In the tables we denote this drawback with the symbol $--$. Conversely, the triangular Shepard interpolant (1) can be applied successfully – without any particular issues and in an efficient way – when the block-based technique is employed.

Finally, to give a better idea of the obtained results, we conclude this section showing for brevity in one case only, i.e. for $n = 40\,000$, and for both Halton and pseudo-random MATLAB data points the reconstructed surfaces of f_1 and f_2 along with the related absolute errors e_i in (6), with $i = 1, \dots, n_e$, see Figures 5–8.

6. Applications

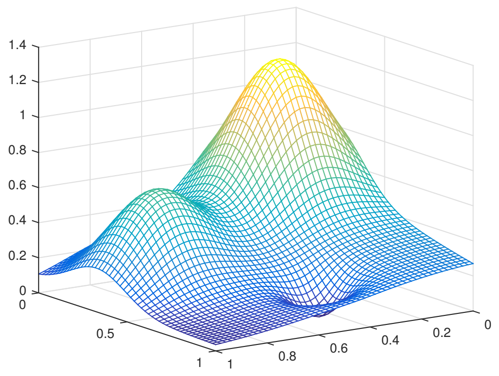
In this section we consider an application of our method to the surface approximation of realistic scattered data in order to show efficacy and accuracy of the proposed technique. In particular, we apply it to the

n	MAE	RMSE	t_{old}	t_{new}
10 000	3.25E − 3	3.03E − 4	26.5909	5.9665
20 000	1.48E − 3	1.45E − 4	294.1082	12.1970
40 000	6.70E − 4	7.48E − 5	−−	26.4157
80 000	4.23E − 4	3.88E − 5	−−	62.3892

Table 2: MAE, RMSE and CPU times computed on Halton points for f_1.

n	MAE	RMSE	t_{old}	t_{new}
10 000	3.84E − 2	4.38E − 3	24.6188	5.9216
20 000	1.59E − 2	2.05E − 3	326.0878	12.2961
40 000	7.47E − 3	1.12E − 3	−−	26.1343
80 000	5.18E − 3	5.30E − 4	−−	57.5480

Table 3: MAE, RMSE and CPU times computed on Halton points for f_2.



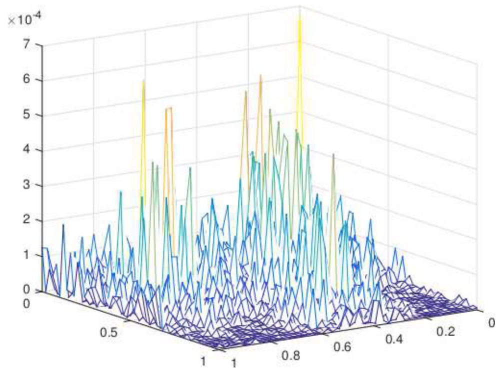


Figure 5: Approximation of Franke's test function (left) and absolute errors (right) obtained by using $n = 40\,000$ Halton points.

reconstruction of a geological surface situated in the Vallée d'Ossau, Pyrénées mountains (France) and to rapidly varying data corresponding to the seafloor surface of one of the deepest parts of the Tonga Trench (Pacific Ocean) [24].

6.1. *Application to the reconstruction of a real surface*

The studied area is located in the Western Pyrénées (30 km south of Pau, Barn, France) in the Vallée d'Ossau which is an old glacial valley. The dataset is constituted of 4 697 functional evaluations, see Figure 9. In order to test the effectiveness of the triangular Shepard method, we extrapolate from the dataset a subset of 97 data to use as evaluation points. We obtain the relative maximum absolute error

$$\mathrm{RMAE} = \max_{1 \le i \le 97} \frac{e_i}{|f(z_i)|} = 3.21\mathrm{E} - 2$$

n	MAE	RMSE	t_{old}	t_{new}
10 000	6.12E − 3	5.24E − 4	26.8241	6.1892
20 000	2.94E − 3	2.65E − 4	248.5067	12.5980
40 000	2.14E − 3	1.51E − 4	−−	26.9432
80 000	9.26E − 4	7.06E − 5	−−	62.5200

Table 4: MAE, RMSE and CPU times computed on pseudo-random MATLAB points for f_1.

n	MAE	RMSE	t_{old}	t_{new}
10 000	5.70E − 2	7.07E − 7	27.9135	7.0858
20 000	2.51E − 2	3.59E − 3	249.8624	12.5749
40 000	1.63E − 2	1.84E − 3	−−	26.9854
80 000	8.28E − 3	8.83E − 4	−−	61.6554

Table 5: MAE, RMSE and CPU times computed on pseudo-random MATLAB points for f_2.

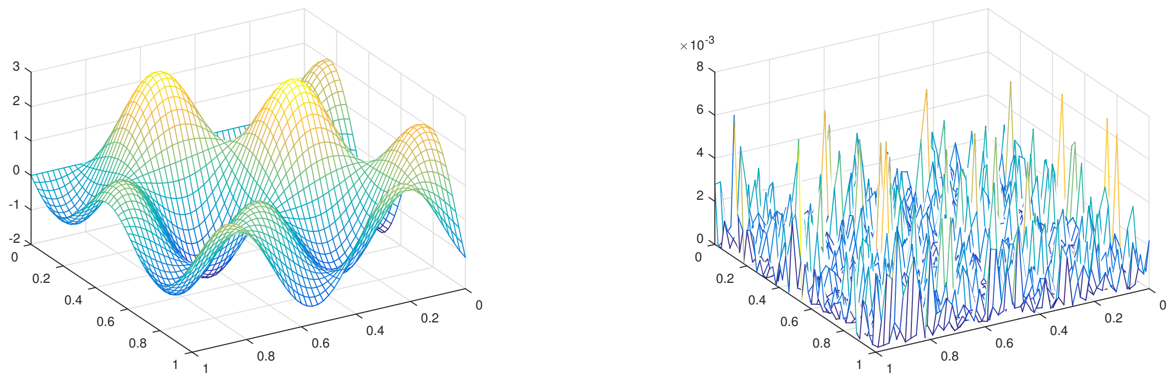


Figure 6: Approximation of test function f_2 (left) and absolute errors (right) obtained by using $n = 40\,000$ Halton points.

and the relative root mean square error

$$\text{RRMSE} = \sqrt{\frac{1}{97}\sum_{i=1}^{97}\left(\frac{e_i}{f(z_i)}\right)^2} = 5.47\text{E} - 4,$$

which demonstrate a very good accuracy of approximation of the triangular Shepard method in reconstructing the real surface. The reconstructed surface coming from geological dataset of Pyrénées is shown in Figure 10.

6.2. Application to the approximation of rapidly varying data

The Tonga Trench is located in the Pacific Ocean and is 10 882 m (35 702 ft) deep at its deepest point, known as Horizon Deep. The trench and its arc form an active subduction zone between two plates of the lithosphere, the Pacific Plate being subducted below the Tonga Plate at the northeastern corner of the Australian Plate. The Tonga Trench spreads in the north-northeast of the Kermadec Islands which are

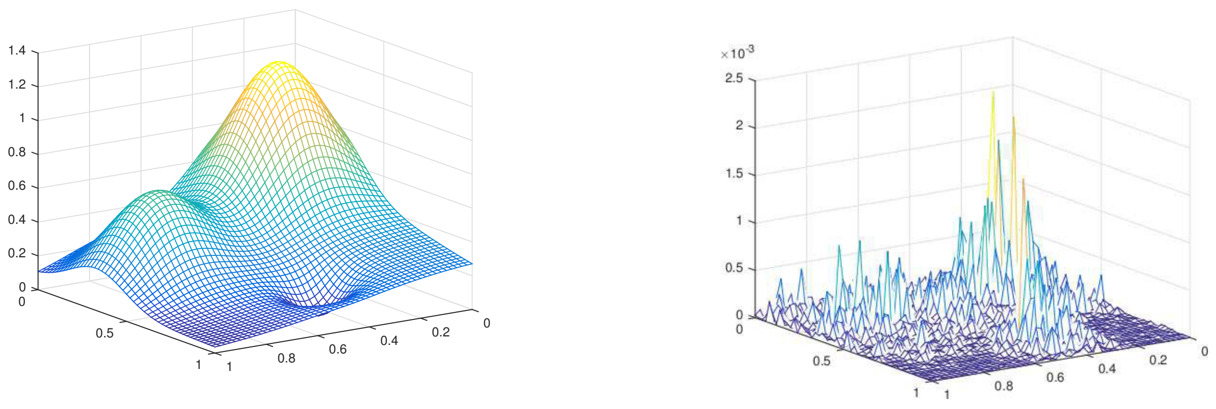


Figure 7: Approximation of Franke's test function (left) and absolute errors (right) obtained by using $n = 40\,000$ pseudo-random MATLAB points.

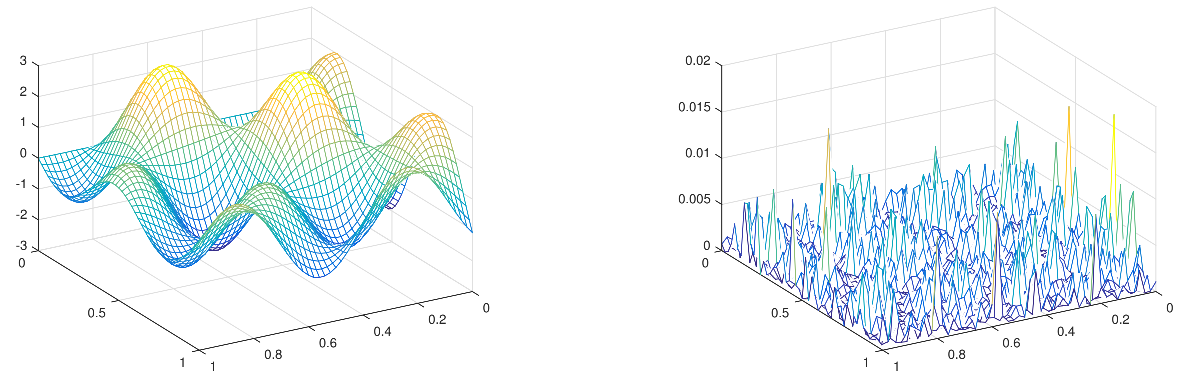


Figure 8: Approximation of test function f_2 (left) and absolute errors (right) obtained by using $n = 40\,000$ pseudo-random MATLAB points.

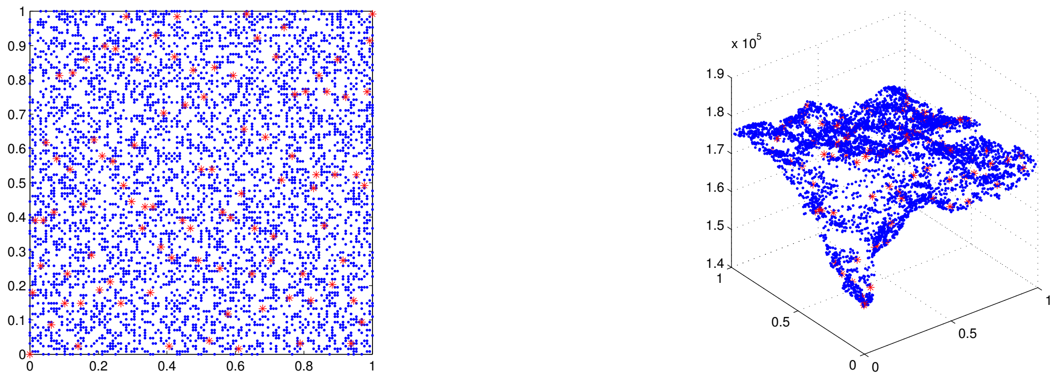


Figure 9: Geological dataset situated in the Vallée d'Ossau, Pyrénées mountains (left) and the corresponding geological surface (right). The 4 600 blue points have been used to reconstruct the surface, the 97 red points have been used as evaluation points to test the effectiveness of the method.

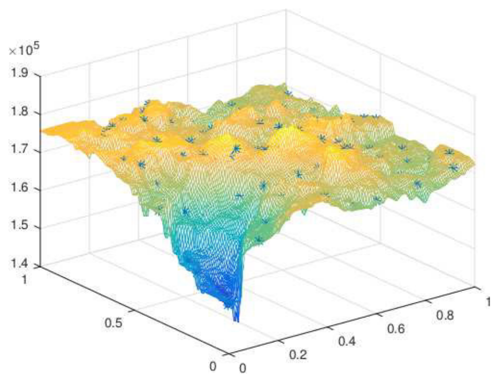


Figure 10: The reconstruction surface of the geological dataset shown in Figure 9 with the extrapolated 97 points (blue stars) evaluated at a regular grid of 150×150 points in the square $[0,1] \times [0,1]$.

situated in the northeast of New Zealand's North Island. The trench turns west north of the Tonga Plate and becomes a transform fault zone. The convergence is taking place at a rate estimated at approximately 15 cm (6 in.) per year; however, recent Global Positioning Satellite measurements indicate in places a convergence of 24 cm (10 in.) per year across the northern Tonga Trench, which is the fastest plate velocity recorded on the planet. Such oceanic trenches are important sites for the formation of what will become continental crust and for the recycling of material back into the mantle. Along the Tonga Trench, mantle-derived melts are transferred to the island arc systems, and abyssal oceanic sediments and fragments of oceanic crust are collected. The Kermadec Trench, to the south, is basically an extension of the Tonga Trench. This zone presents large variations and is of great interest to study. In our example we focus on the deepest part of the Tonga Trench. The considered dataset is constituted of 3 223 functional evaluations, see Figure 11 (left). In order to test the effectiveness of the triangular Shepard method we randomly extrapolate, from the given dataset, a subset of 67 data, displayed in red in Figure 11 (left), to use as evaluation points. We obtain the relative maximum absolute error

$$\text{RMAE} = \max_{1 \le i \le 67} \frac{e_i}{|f(z_i)|} = 8.38\text{E} - 2$$

and the relative root mean square error

$$\text{RRMSE} = \sqrt{\frac{1}{67} \sum_{i=1}^{67} \left(\frac{e_i}{f(z_i)} \right)^2} = 1.95\text{E} - 3,$$

which demonstrate a very good accuracy of approximation of the triangular Shepard method in reconstructing the real surface. In order to test the reproduction quality of the triangular Shepard method we run the algorithm on a regular grid of 100×100 evaluation points in $[184.02, 186.02] \times [-26, -22]$. The obtained compact triangulation is displayed in Figure 11 (right) and the resulting surface is displayed in Figure 12, which points out that an excellent reconstruction result is achieved by the proposed algorithm.

7. Conclusions and future work

In this paper we presented a new interpolation algorithm for fast computing triangular Shepard interpolants. Since the triangle-based approach requires to identify useful triangulations associated with the dataset, we proposed the use of a versatile block-based partitioning structure and related searching procedure, which turn out to be particularly efficient from a computational viewpoint. In fact, as evident from numerical experiments and applications considered in this work, our interpolation scheme enables to quickly deal with a large number of points, while standard routines cannot solve successfully such approximation problems.

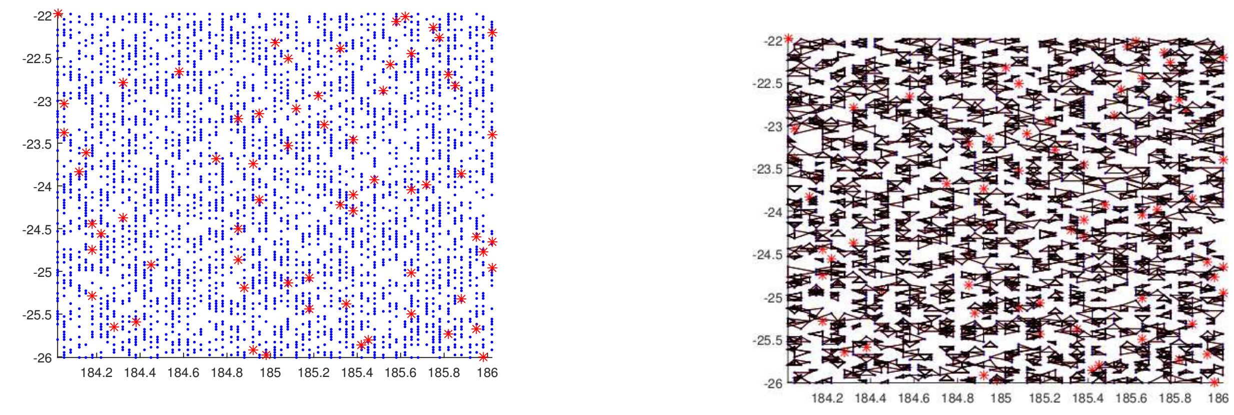


Figure 11: Geological dataset situated in Tonga Trench, Pacific Ocean (left) and the corresponding compact triangulation (right).

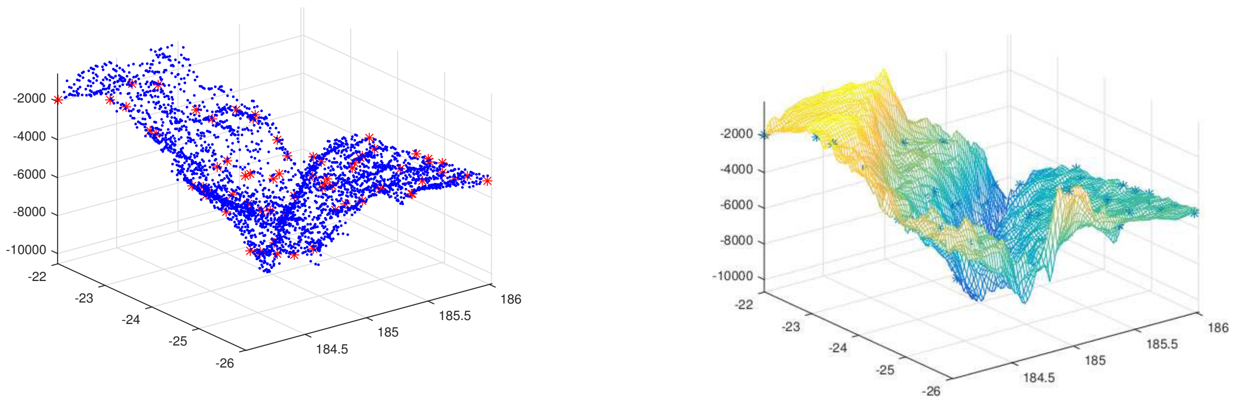


Figure 12: The reconstruction of the geological surface at the extrapolated 67 red points (right). The 3 156 blue points have been used to reconstruct the surface, the 67 red points have been used as evaluation points to test the effectiveness of the method. The reconstruction surface of the geological dataset with the extrapolated 67 points (blue stars) evaluated at a regular grid of 100×100 points in the square $[184.02, 186.02] \times [-26, -22]$ (left).

As future work we propose to extend the triangular Shepard method to the 3D case, possibly studying new interpolation schemes to be applied on various domains such as sphere and other manifolds, see e.g. [25].

Acknowledgements

This research has been accomplished within the RITA “Research ITalian network on Approximation”. This work was partially supported by the 2016-2017 projects “Metodi numerici e computazionali per le scienze applicate” and “Approssimazione multivariata e algoritmi efficienti con applicazioni a problemi algebrici, differenziali e integrali” of the Department of Mathematics of the University of Torino. Moreover, this research was supported by GNCS–INdAM 2016-2017 project and by a research fellow of the Centro Universitario Cattolico.

References

[1] F. Dell’Accio, F. Di Tommaso, K. Hormann, On the approximation order of triangular Shepard interpolation, IMA J. Numer. Anal. 36 (2016) 359–379.

[2] R. Cavoretto, A. De Rossi, E. Perracchione, Efficient computation of partition of unity interpolants through a block-based searching technique, Comput. Math. Appl. 71 (2016) 2568–2584.

[3] R. Cavoretto, S. De Marchi, A. De Rossi, E. Perracchione, G. Santin, Partition of unity interpolation using stable kernel-based techniques, Appl. Numer. Math. 116 (2017) 95–107.

[4] D. Shepard, A two-dimensional interpolation function for irregularly-spaced data, in: Proceedings of the 1968 23rd ACM National Conference, ACM '68, ACM, New York, NY, USA, 1968, pp. 517–524.

[5] G. E. Fasshauer, Meshfree Approximation Methods with Matlab, World Scientific Publishing Co., Inc., Singapore, 2007.

[6] G. Allasia, R. Besenghi, R. Cavoretto, A. De Rossi, Scattered and track data interpolation using an efficient strip searching procedure, Appl. Math. Comput. 217 (2011) 5949–5966.

[7] G. Fasshauer, M. McCourt, Kernel-based Approximation Methods using Matlab, World Scientific Publishing Co., Inc., Singapore, 2015.

[8] F. Dell'Accio, F. Di Tommaso, Scattered data interpolation by Shepard's like methods: Classical results and recent advances, Dolomites Res. Notes Approx. 9 (2016) 32–44.

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

[10] H. Wendland, Scattered Data Approximation, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2005.

[11] W. I. Thacker, J. Zhang, L. T. Watson, J. B. Birch, M. A. Iyer, M. W. Berry, Algorithm 905: SHEPPACK: Modified Shepard algorithm for interpolation of scattered multivariate data, ACM Trans. Math. Softw. 37 (2010) 1–20.

[12] R. Caira, F. Dell'Accio, F. Di Tommaso, On the bivariate Shepard-Lidstone operators, J. Comput. Appl. Math. 236 (2012) 1691–1707.

[13] G. Coman, Shepard operators of Birkhoff-type, Calcolo 35 (1998) 197–203.

[14] F. Costabile, F. Dell'Accio, F. Di Tommaso, Enhancing the approximation order of local Shepard operators by Hermite polynomials, Comput. Math. Appl. 64 (2012) 3641–3655.

[15] F. A. Costabile, F. Dell'Accio, F. Di Tommaso, Complementary Lidstone interpolation on scattered data sets, Numer. Algorithms 64 (2013) 157–180.

[16] F. Dell'Accio, F. Di Tommaso, Complete Hermite-Birkhoff interpolation on scattered data by combined Shepard operators, J. Comput. Appl. Math. 300 (2016) 192–206.

[17] F. F. Little, Convex combination surfaces, in: R. E. Barnhill, W. Boehm (Eds.), Surfaces in Computer Aided Geometric Design, volume 1479, North-Holland, 1983, pp. 99–108.

[18] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, A. Y. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, J. ACM 45 (1998) 891–923.

[19] R. Farwig, Rate of convergence of Shepard's global interpolation formula, Math. Comp. 46 (1986) 577–590.

[20] T.-T. Wong, W.-S. Luk, P.-A. Heng, Sampling with Hammersley and Halton points, J. Graph. Tools 2 (1997) 9–24.

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

[22] R. Cavoretto, A. De Rossi, E. Perracchione, Optimal selection of local approximants in RBF-PU interpolation, J. Sci. Comput. (2017) in press.

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

[24] C. Gout, C. Le Guyader, L. Romani, A.-G. Saint-Guirons, Approximation of surfaces with fault(s) and/or rapidly varying data, using a segmentation process, D^m-splines and the finite element method, Numer. Algorithms 48 (2008) 67–92.

[25] G. Allasia, R. Cavoretto, A. De Rossi, Hermite-Birkhoff interpolation on scattered data on the sphere and other manifolds, Appl. Math. Comput. (2017) in press.