J. Approx. Softw., Vol. 1, No. 2 (2024) 4

Journal of Approximation Software

The Multinode Shepard Method: MATLAB Implementation

F. Dell'Accio [1], F. Di Tommaso [1,⋆], and F. Larosa [1]

[1] *Department of Mathematics and Computer Science, University of Calabria*

Received: 12/07/2024 – Published: 06/09/2024
Communicated by: R. Cavoretto

Abstract

The multinode Shepard method is an extension of inverse distance weighting, developed as a generalization of the triangular Shepard method to further improve interpolation accuracy in situations where the classic Shepard method results are limited. In particular, it considers multiple nodes for local interpolation, offering greater flexibility and improved accuracy in estimates. In this paper, we present two algorithms for computing the multinode Shepard interpolant, providing the related pseudocodes and MATLAB implementations.

Keywords: Multivariate Lagrange interpolation, Multinode Shepard method, Rational Interpolant, (MSC2020: 65D05, 41A05, 41A20)

1 Introduction

The approximation of scattered data is a crucial technique in modern science and engineering, enabling the extraction of significant information from incomplete or irregularly distributed data sets. One method for interpolating such data is Shepard's method.

The classical Shepard operator reconstructs a function through a weighted combination of its values at data points. The weights are the normalization of the inverse distances from the approximation point to the nodes. The nodes are scattered, that is, without any particular structure. More precisely, let $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ be a set of nodes in $\mathbb{R}^2$, and $f : \mathbb{R}^2 \to \mathbb{R}$ the function for which only the evaluations $f_i = f(\boldsymbol{x}_i)$ at the nodes are known. The Shepard operator [15] is defined as

$$S_\mu[f](\boldsymbol{x}) = \sum_{i=1}^{n} A_{\mu,i}(\boldsymbol{x}) f_i, \qquad \boldsymbol{x} \in \mathbb{R}^2, \tag{1}$$

⋆ Corresponding author: filomena.ditommaso@unical.it

where

$$A_{\mu,i}(\boldsymbol{x}) = \frac{\|\boldsymbol{x}-\boldsymbol{x}_i\|^{-\mu}}{\sum_{j=1}^{n} \|\boldsymbol{x}-\boldsymbol{x}_j\|^{-\mu}}, \qquad \boldsymbol{x} \in \mathbb{R}^2, \tag{2}$$

where $\mu > 0$ is a control parameter and $\|\cdot\|$ denotes the Euclidean norm.

The Shepard operator interpolates the data f_i at the nodes $\boldsymbol{x}_i$, even though it presents, for $u > 1$, flat spots in the neighborhood of all data point (which become cusps if $u \leq 1$). Moreover, it has a reproduction degree of zero, meaning it exclusively reproduces constant polynomials. This last property strongly influences the accuracy of the approximation provided by the Shepard interpolant, which converges to the function f with at most linear speed [13]. In order to overcome these limitations, several modifications have been proposed over the years [4].

The first significant modification of the Shepard operator is due to Little [14], who, based on the general idea of defining interpolants through convex combinations, proposed the following improvement. He considered a triangulation $T = \{t_j\}_{j=1}^{s}$ of the set X and replaced the values f_i in the expression of the Shepard operator (1) with the evaluations $L_j(\boldsymbol{x})$ of the linear polynomials interpolating at the vertices of each triangle t_j. Furthermore, he replaced the classical weight functions (2), obtained from the normalization of the inverse distances from individual nodes $\boldsymbol{x}_i$, with the product of the normalized inverse distances from the vertices $\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}$ of each triangle t_j. The resulting operator, known as the triangular Shepard operator, interpolates the data f_i at the nodes $\boldsymbol{x}_i$ and reproduces the polynomials up to degree 1. The formal definition is as follows

$$K_\mu[f](\boldsymbol{x}) = \sum_{j=1}^{s} B_{\mu,j}(\boldsymbol{x}) L_j(\boldsymbol{x}), \qquad \boldsymbol{x} \in \mathbb{R}^2, \tag{3}$$

where

$$B_{\mu,j}(\boldsymbol{x}) = \frac{\prod_{l=1}^{3} \|\boldsymbol{x}-\boldsymbol{x}_{j_l}\|^{-\mu}}{\sum_{k=1}^{s} \prod_{l=1}^{3} \|\boldsymbol{x}-\boldsymbol{x}_{k_l}\|^{-\mu}}, \qquad \boldsymbol{x} \in \mathbb{R}^2. \tag{4}$$

This deep modification of the Shepard operator aims to improve the accuracy and robustness of scattered data interpolation. The properties of the triangular Shepard operator have been extensively studied in [7]. In particular, the quadratic convergence of the triangular Shepard interpolant to the function f has been demonstrated, both in the case of regular triangulations, such as Delaunay, and in the case of compact triangulations, which do not exclude the intersection of triangles in common parts that are not only vertices or adjacent sides. This latter observation has paved the way for the not easy generalization of the classical Shepard operator, aiming to further increase the polynomial precision and, consequently, the accuracy of the approximation. This generalization, known as the multinode Shepard operator, has been proposed in a series of papers first dealing with the hexagonal case in $\mathbb{R}^2$ [6] and the tetrahedral case in $\mathbb{R}^3$ [3], relying on local barycentric coordinate systems, and then the general case in [8], exploiting a new representation of the local interpolation polynomial in Taylor form, centered at the barycenter of the local system of interpolation nodes [9].

In this paper, we present and discuss two algorithms for the computation of the multinode Shepard method. The paper is organized as follows. In Section 2 we recall the definition of the multinode Shepard operator and describe its properties. In Section 2.1 we present two algorithms for the implementation of the multinode Shepard method. In section 2.2 we give the Matlab implementation of the two codes. In Section 2.3 we report some numerical experiments

to test the effectiveness of the proposed algorithms. Finally, in Section 3 we summarize the benefits of the multinode Shepard method in terms of accuracy and computational efficiency.

2 Multinode Shepard method

Let $\Omega \subset \mathbb{R}^d$, $d \geq 2$, a non-empty connected open set, $\partial\Omega$ its boundary, $X = \{\boldsymbol{x}_i\}_{i=1}^n \subset \Omega \cup \partial\Omega$ a finite set of pairwise distinct scattered nodes and $f = \{f_i\}_{i=1}^n$ a set of function values associated to X. Let $r \in \mathbb{N}$ and $m = \binom{r+d}{d} = \dim\left(\mathscr{P}_r(\mathbb{R}^d)\right)$, where $\mathscr{P}_r(\mathbb{R}^d)$ denotes the space of polynomials of d variables of total degree $\leq r$.

We assume that a set $\{\sigma_j\}_{j=1}^s$ is given, such that for each $j = 1,\ldots,s$, $\sigma_j = \{\boldsymbol{x}_{j_k}\}_{k=1}^m \subset X$ is unisolvent for the polynomial space $\mathscr{P}_r(\mathbb{R}^d)$ and

$$\bigcup_{j=1}^{s} \sigma_j = X \tag{5}$$

(to shorten the notation, for each $j = 1,\ldots,s$, we are denoting with $j_k = \varphi_j(k)$ the image of $k \in \{1,\ldots,m\}$ by an injective map φ_j from $\{1,\ldots,m\}$ into $\{1,\ldots,n\}$).

A convenient way to represent the unique polynomial $P_j \in \mathscr{P}_r(\mathbb{R}^d)$, $j = 1,\ldots,s$, interpolating on $\sigma_j = \{\boldsymbol{x}_{j_1},\ldots,\boldsymbol{x}_{j_m}\}$ the data $\{f_{j_1},\ldots,f_{j_m}\}$ is given by the

$$P_j(\boldsymbol{x}) = \sum_{k=1}^{m} \ell_{j,k}(\boldsymbol{x}) f_{j_k}, \qquad \boldsymbol{x} \in \mathbb{R}^d,$$

where

$$\ell_{j,k}(\boldsymbol{x}) = \sum_{|\alpha| \leq r} a_{\alpha}^{(j,k)} \left(\boldsymbol{x} - \boldsymbol{x}_j^{(b)}\right)^{\alpha} \tag{6}$$

are the Lagrange fundamental polynomials written in the Taylor basis centered at the barycenter $\boldsymbol{x}_j^{(b)}$ of σ_j and $\alpha \in \mathbb{N}^m \cup \{(0,\ldots,0)\}$ is a multi-index (for more details see [9]). As well-known, the fundamental Lagrange polynomials satisfy the Kronecker delta property

$$\ell_{j,k}\left(\boldsymbol{x}_{j_l}\right) = \delta_{kl}, \quad j = 1,\ldots,s; \quad k,l = 1,\ldots,m. \tag{7}$$

The multinode inverse distance weighted functions based on the covering $\{\sigma_j\}_{j=1}^s$ are defined as follows [5]

$$W_{\mu,j}(\boldsymbol{x}) = \frac{\prod_{k=1}^{m} \left\|\boldsymbol{x} - \boldsymbol{x}_{j_k}\right\|^{-\mu}}{\sum_{l=1}^{s} \prod_{\lambda=1}^{m} \left\|\boldsymbol{x} - \boldsymbol{x}_{l_\lambda}\right\|^{-\mu}}, \qquad j = 1,\ldots,s, \quad \mu > 0. \tag{8}$$

The multinode functions form a partition of unity

$$\sum_{j=1}^{s} W_{\mu,j}(\boldsymbol{x}) = 1, \quad \boldsymbol{x} \in \mathbb{R}^d, \tag{9}$$

and satisfy the following interpolation properties

$$W_{\mu,j}(\boldsymbol{x}_i) = 0 \text{ for all } \boldsymbol{x}_i \notin \sigma_j, \quad \sum_{j \in \mathscr{J}_i} W_{\mu,j}(\boldsymbol{x}_i) = 1, \tag{10}$$

where we set

$$\mathscr{J}_i = \{j \in \{1,\ldots,s\} : \boldsymbol{x}_i \in \sigma_j\}, \quad i = 1,\ldots,n.$$

In addition, if $\mu > 2$ the multinode functions $W_{\mu,j}(\boldsymbol{x})$ satisfy the following differential properties

$$\nabla W_{\mu,j}(\boldsymbol{x}_i) = \boldsymbol{0} \text{ for all } \boldsymbol{x}_i \notin \sigma_j, \quad \sum_{j\in\mathscr{J}_i} \nabla W_{\mu,j}(\boldsymbol{x}_i) = \boldsymbol{0}, \tag{11}$$

and

$$HW_{\mu,j}(\boldsymbol{x}_i) = \boldsymbol{0} \text{ for all } \boldsymbol{x}_i \notin \sigma_j, \quad \sum_{j\in\mathscr{J}_i} HW_{\mu,j}(\boldsymbol{x}_i) = \boldsymbol{0}, \tag{12}$$

where, as usual, $\nabla W_{\mu,j}(\boldsymbol{x})$ and $HW_{\mu,j}(\boldsymbol{x})$ denote the gradient and the Hessian matrix of $W_{\mu,j}(\boldsymbol{x})$, respectively. Finally, they are rational functions without real singularities if μ is an even integer (for more details, see [5]).

The multinode Shepard operator is a blend of the local interpolation polynomials realized by using multinode functions as follows

$$\mathscr{M}_\mu[f](\boldsymbol{x}) = \sum_{j=1}^{s} W_{\mu,j}(\boldsymbol{x}) P_j(\boldsymbol{x}) = \frac{\sum_{j=1}^{s}\prod_{k=1}^{m} \left\|\boldsymbol{x}-\boldsymbol{x}_{j_k}\right\|^{-\mu} P_j(\boldsymbol{x})}{\sum_{j=1}^{s}\prod_{k=1}^{m} \left\|\boldsymbol{x}-\boldsymbol{x}_{j_k}\right\|^{-\mu}}. \tag{13}$$

Since the property (9) $\mathscr{M}_\mu[\cdot]$ reproduces polynomials of d variables of total degree $\leq r$, while (10) imply that $\mathscr{M}_\mu[f]$ interpolates data f_i at $\boldsymbol{x}_i$, $i = 1,\ldots,n$. Moreover, by assuming that the set X is contained in a compact convex domain Ω and that the function f is of class $C^r(\Omega)$ with partial derivatives Lipschitz-continuous of order r, as proven in [5, Theorem 3.1], the multinode Shepard operator has an approximation accuracy of $O(h^{p+1})$ for each $\mu > \frac{d+p+1}{m}$. Here h denotes the fill distance of the set X.

From equation (13) it follows that the multinode Shepard operator is not uniquely defined, depending on the particular covering $\{\sigma_j\}_{j=1}^{s}$ of X. The existence of such a covering is *almost surely* guaranteed [11]. A straightforward determination of $\{\sigma_j\}_{j=1}^{s}$ can be done by considering, for each scattered point $\boldsymbol{x}_i$, the set of $m+q$, $q > 0$, nearest points and by choosing, among them, the subset of m points for which the local approximation to the function $f(\boldsymbol{x})$ provided by the polynomial $P_j(\boldsymbol{x})$ is near to the optimal one [10]. For $i \neq i'$, by denoting with $\sigma(\boldsymbol{x}_i)$ and $\sigma(\boldsymbol{x}_{i'})$ the subsets determined starting from the interpolation nodes $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$, respectively, we have $\sigma_j = \sigma(\boldsymbol{x}_i)$ and $\sigma_{j'} = \sigma(\boldsymbol{x}_{i'})$, $j \neq j'$, if $\sigma(\boldsymbol{x}_i) \neq \sigma(\boldsymbol{x}_{i'})$, otherwise we set $\sigma_j = \sigma(\boldsymbol{x}_i) = \sigma(\boldsymbol{x}_{i'})$. While ensuring the covering condition, this procedure leads to an overly expensive definition of the multinode Shepard operator, due to the large number of terms in the sum (13).

In the following, we specialize in the case $d = 2$. Algorithms for general dimensions $d > 2$ can be obtained using the given approaches.

2.1 Pseudo-codes of two algorithms for the implementation of the multinode Shepard method

In this section, we present two algorithms for the computation of the multinode Shepard method. The Algorithm 1, based on the procedure mentioned above, chooses in the set of $m+q$, $q > 0$, nearby nodes to $\boldsymbol{x}_i$, the m discrete Leja points computed by the algorithm presented in [1]. This algorithm is based on the $PA = LU$ factorization of the Gram matrix. The set σ_j consists of the

points *related* to the first m rows after the $PA = LU$ factorization. Gaussian elimination with row pivoting performs a greedy optimization (i.e. not precise but still valid) of the Vandermonde determinant, iteratively searching for the new row (i.e. selecting the new interpolation point) such that the modulus of the increased determinant is maximized.

To speed up Algorithm 1 and reduce its computational cost, we introduce Algorithm 2. The key point of Algorithm 2 is to drastically reduce the number of subsets σ_j. Specifically, we create a copy $X' = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ of the node set X, whose first node is $\boldsymbol{x}_1$. In the process of determination of the covering $\{\sigma_j\}_{j=1}^{s}$, the set X remains fixed and is used to determine the subset N_i of the nearby nodes to $\boldsymbol{x}_i$. We reorder the nodes of N_i according to their increasing distances from $\boldsymbol{x}_i$, with $\boldsymbol{x}_i$ being the first node of X'. At the j-th step this rearrangement allows the identification of the subset σ_j by using the procedure stated in Algorithm 1, that is through the $PA = LU$ factorization of the Gram matrix. The subset σ_j is then subtracted from X', by maintaining the initial order of the nodes. Since at the step $j+1$ the new set $X' = X' \setminus \sigma_j$ will no longer contain $\boldsymbol{x}_i$ but a new first node $\boldsymbol{x}_{i'}$, the procedure ends when X' is empty.

To determine N_i, we set $M(\Omega)$ the Lebesgue measure of Ω, $\ell = \sqrt{\frac{\binom{r+1+2}{2} M(\Omega)}{n}}$ and $Q_i(\rho) = \left[x_i - \frac{\rho}{2}, x_i + \frac{\rho}{2}\right] \times \left[y_i - \frac{\rho}{2}, y_i + \frac{\rho}{2}\right]$, $\rho > 0$. Then

$$N_i = X \cap Q_i(\ell(1 + k/10)) \tag{14}$$

where k is the first non-negative integer such that $\#(N_i) \geq \binom{r+1+2}{2} = \dim\left(\mathscr{P}_{r+1}(\mathbb{R}^2)\right)$.

Algorithm 1 The multinode Shepard method with a m-tuple associated to each $\boldsymbol{x}_i$

Require: $X = \{\boldsymbol{x}_i\}_{i=1}^{n}$, the set of scattered points
$f = \{f_i\}_{i=1}^{n}$, the set of function values
r, the degree of the local polynomial interpolants
q, with $m+q$ the number of nearest neighbour points for determining $\{\sigma_j\}_{j=1}^{s}$
μ, the power parameter for the computation of (8)
$\boldsymbol{x}$, the set of approximation points
Ensure: $f(\boldsymbol{x})$, approximations of the function f at the points of $\boldsymbol{x}$

1: Set *Num*, the numerator of the multinode Shepard operator (13), equal to 0
2: Set *Den*, the denominator of the multinode Shepard operator (13), equal to 0
for $i = 1, \dots, n$ **do**
  3.1: Compute the vector *Dn* containing the distances between the i-th node and all other nodes
  3.2: Sort *Dn* according to the increasing distances from $\boldsymbol{x}_i$
  3.3: Consider the $m+q$ nearest points to $\boldsymbol{x}_i$ and choose, among them, the subset σ_j of m discrete Leja points by using the algorithm proposed in [1]
  if the subset σ_j has not been yet considered **then**
    4.1: Compute the polynomial interpolant of f based on the points of σ_j
    4.2: Compute the j-th term of the numerator and of the denominator of the multinode operator (13) and sum them up to *Num* and *Den*, respectively
  end if
end for
5: Compute the multinode Shepard operator as the ratio between *Num* and *Den*

Algorithm 2 The multinode Shepard method with minimized number of m-tuples

Require: $X = \{\boldsymbol{x}_i\}_{i=1}^n$, the set of scattered points
$f = \{f_i\}_{i=1}^n$, the set of function values
r, the degree of the local polynomial interpolants
$\boldsymbol{x}$, the set of approximation points
Ensure: $f(\boldsymbol{x})$, approximations of the function f at the points of $\boldsymbol{x}$

1: Compute $\ell = \sqrt{\frac{\binom{r+1+2}{2} M(\Omega)}{n}}$, the length of the side of the square to determine the set of nearby points
2: Create a copy X' of X.
3: Set *Num*, the numerator of the multinode Shepard operator (13), equal to 0
4: Set *Den*, the denominator of the multinode Shepard operator (13), equal to 0
5: Set i equal to 0
while $X' \neq \emptyset$ **do**
  6.1: Compute the set $N_i = X \cap Q_i(\ell)$
  6.2: Set k equal to 1
  while $\#(N_i) < \binom{r+1+2}{2}$ **do**
    7.1: Compute the set $N_i = X \cap Q_i(\ell(1+k/10))$
    7.2: Update k to $k+1$
  end while
  8: Sort N_i according to the increasing distances between the first node $\boldsymbol{x}'_i$ of X' and all nodes in N_i
  9: Choose the among the points in the ordered N_i, the subset σ_j of m discrete Leja points by using the algorithm proposed in [1]
  if the subset σ_j has not been yet considered **then**
    10.1: Compute the polynomial interpolant of f based on the points of σ_j
    10.2: Compute the j-th term of the numerator and of the denominator of the multinode operator (13) and sum them up to *Num* and *Den*, respectively
    11: Update i to $i+1$
  end if
end while
12: Compute the multinode Shepard operator as the ratio between *Num* and *Den*

2.2 MATLAB code

In this section we describe the MATLAB functions and demos available collected in a package named `Multinode_Shepard` freely available at [12].

2.2.1 Function `Multinode_Shepard1.m`

The function `Multinode_Shepard1.m` implements the Multinode Shepard method as described by the pseudocode Algorithm 1. In particular, we have the following Input and Output arguments:

INPUT:

- xn (double array): vector of the x-coordinates of the nodes
- yn (double array): vector of the y-coordinates of the nodes
- fn (double array): vector of the function values at the nodes (xn,yn)
- r (integer scalar): degree of the local polynomial interpolant
- q (integer scalar): number of additional points to select the m-tuple σ_j
- mu (integer scalar): power parameter
- x (double array): vector of the x-coordinate of the evalutation points
- y (double array): vector of the y-coordinate of the evalutation points

OUTPUT:

- MO (double array): values of the multinode Shepard operator at the points (x,y)
- s (integer scalar): number of the m-tuples σ_j

The function `Multinode_Shepard1.m` makes use of the following auxiliary functions and MATLAB functions:

- `powers`: function which computes the powers of the bivariate monomial basis of total degree d by using the routine mono_next_grlex.m
- `length`: computes the length of a vector
- `sort`: sorts in ascending order the elements of v, where v is the input vector and returns also the sort index I which specifies how the elements of v were rearranged to obtain the sorted output vector
- `ismember`: checks if a vector has the same entrances of the row of a matrix
- `sum`: computes the sum of the elements of a vector
- `BivVand`: function which computes the bivariate Vandermonde matrix
- `lu`: computes the LU factorization of a matrix and returns unit lower triangular matrix L, upper triangular matrix U, and permutation matrix P

- `backslash`: computes the solution of the linear system $Ax = b$, where A is a matrix and b is the known term
- `eps`: spacing of floating point numbers
- `prod`: computes the product of the over each column of a matrix

2.2.2 Function `Multinode_Shepard2.m`

The function `Multinode_Shepard2.m` implements the Multinode Shepard method as described by the pseudocode Algorithm 2. The output arguments are the same as the function `Multinode_Shepard1.m` described in Section 2.2.1 with the addition of the use of the MATLAB function `isempty.m` which allows to establish if a vector is empty. The input arguments are:

INPUT:

- xn (double array): vector of the x-coordinates of the nodes
- yn (double array): vector of the y-coordinates of the nodes
- fn (double array): vector of the function values at the nodes (xn,yn)
- r (integer scalar): degree of the local polynomial interpolant
- mu (integer scalar): power parameter
- x (double array): vector of the x-coordinate of the evaluation points
- y (double array): vector of the y-coordinate of the evaluation points

2.2.3 Demos

The demo `demo_trial` illustrates the numerical experiments provided in Section 2.4. The demo `demo_Stromboli` illustrates the numerical experiments provided in Section 2.3 by selecting, through a menu, the degree r of the local polynomial interpolant.

2.3 Numerical experiments

In this section we present some numerical experiments in order to show the efficiency of the multinode Shepard method. All the tests have been performed on a laptop with a 11th Gen Intel(R) Core(TM) i7-1165G7 2.80GHz 1.69 GHz processor and 16.00 GB RAM.

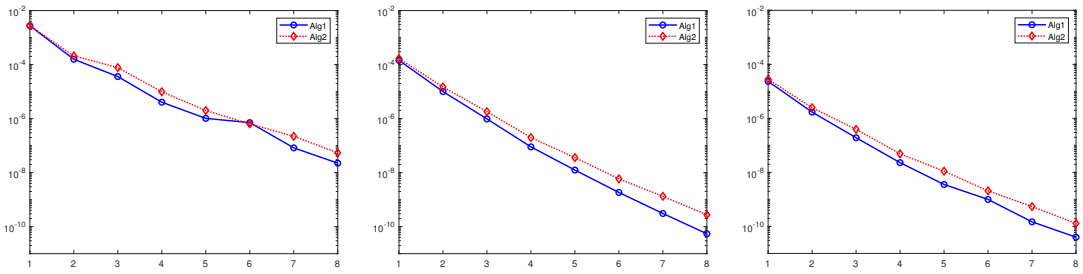
2.4 Reconstrution of smooth surfaces

For the first series of experiments, we make use of a set of 10000 Halton interpolation points and we use them to approximate the Franke function

$$\begin{aligned} f_1(x,y) = \; & 0.75e^{-\frac{(9x-2)^2+(9y-2)^2}{4}} + 0.50e^{-\frac{(9x-7)^2+(9y-3)^2}{4}} \\ & +0.75e^{-\frac{(9x+1)^2}{49}-\frac{(9y+1)^2}{10}} - 0.20e^{-(9x-4)^2-(9y-7)^2} \end{aligned}$$

in the unit square $[0,1]\times[0,1]$. We compute the pointwise error

$$e_i = |f(\boldsymbol{x}_i^*) - \mathscr{M}_4[f](\boldsymbol{x}_i^*)|,$$



MAE *MEAE* *RMSE*

Figure 1: Semilog plot of the maximum absolute errors (*MAE*), mean absolute errors (*MEAE*) and root mean square absolute errors (*RSME*) by varying the degree r of the local polynomial approximant from 1 to 8.

	number of m-tuples		CPU time (sec)	
r	Alg. 1	Alg. 2	Alg. 1	Alg. 2
1	9890	5173	7	3
2	10000	3354	13	3
3	10000	2406	15	3
4	9999	1783	31	4
5	10000	1362	44	4
6	10000	1183	58	4
7	10000	947	74	4
8	10000	802	88	4

Table 1: Number of m-tuples and CPU time (in seconds) for the experiments in Figure 1.

at a regular grid of $n_e = 100 \times 100$ points $\boldsymbol{x}_i^* \in [0,1] \times [0,1]$. We compare Algorithm 1 and Algorithm 2 by computing the maximum, mean and root mean square absolute errors

$$MAE = \max_{1 \le i \le ne} e_i; \quad MEAE = \frac{1}{ne} \sum_{i=1}^{ne} e_i; \quad RMSE = \sqrt{\frac{1}{ne} \sum_{i=1}^{ne} e_i^2} \tag{15}$$

and the CPU time (in seconds) for the computation of the approximate surface. In Figure 1 we display the semilog plot of the errors for the two algorithms by varying the degree of the local polynomial interpolant from 1 to 8 and in Table 1 we report the associated number s of m-tuples.

2.5 Reconstruction of surfaces from real-world data

For the second series of experiments, we make use of a set of real-world data related to the Stromboli Volcano including the Sciara del Fuoco of 2002 lava flow [2] which is constituted by 422710 DEM (Digital Elevation Model) data from which we extract the set of 97020 mock-Halton data. More precisely, we consider the set of 100000 Halton data mapped into the rectangle $[0,2060] \times [0,818]$ and we extract from the DEM data the ones that are closer to the Halton data by discarding the duplicates. To test the effectiveness of the multinode Shepard method, we extract from the mock-Halton dataset a subset $Z_{ne} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{ne}\}$ of $ne = 2000$ data to use as evaluation points.

In Table 2 we compare Algorithm 1 and Algorithm 2 by computing the maximum, mean and root mean square absolute errors as in (15) and in Table 3 we compare the maximum, mean

	MAE		*MEAE*		*RMSE*	
r	Alg. 1	Alg. 2	Alg. 1	Alg. 2	Alg. 1	Alg. 2
1	$1.03e+1$	$7.90e+0$	$2.73e-1$	$2.90e-1$	$1.34e-2$	$1.38e-2$
2	$7.08e+0$	$2.52e+1$	$2.39e-1$	$2.97e-1$	$1.14e-2$	$2.02e-2$
3	$1.23e+1$	$1.15e+1$	$2.64e-1$	$3.61e-1$	$1.36e-2$	$1.96e-2$
4	$7.28e+0$	$2.21e+1$	$2.37e-1$	$3.42e-1$	$1.16e-2$	$2.00e-2$
5	$1.05e+1$	$1.62e+1$	$2.87e-1$	$4.19e-1$	$1.45e-2$	$2.23e-2$
6	$1.45e+1$	$2.49e+1$	$2.88e-1$	$3.97e-1$	$1.50e-2$	$2.27e-2$

Table 2: Maximum, mean and root mean square absolute errors.

	MAE_{rel}		$MEAE_{rel}$		$RMSE_{rel}$	
r	Alg. 1	Alg. 2	Alg. 1	Alg. 2	Alg. 1	Alg. 2
1	$2.28e-1$	$1.90e-1$	$1.68e-3$	$1.70e-3$	$1.74e-4$	$1.69e-4$
2	$1.24e-1$	$1.72e+0$	$1.45e-3$	$2.55e-3$	$1.38e-4$	$8.74e-4$
3	$7.60e-1$	$1.46e+0$	$2.04e-3$	$2.79e-3$	$4.55e-4$	$7.62e-4$
4	$3.29e-1$	$5.67e-1$	$1.51e-3$	$2.47e-3$	$2.11e-4$	$4.02e-4$
5	$1.94e-1$	$5.07e-1$	$1.78e-3$	$2.75e-3$	$2.03e-4$	$3.95e-4$
6	$3.07e-1$	$2.18e-1$	$1.71e-3$	$2.15e-3$	$2.16e-4$	$2.02e-4$

Table 3: Maximum, mean and root mean square relative errors.

and root mean square relative errors

$$MAE_{rel} = \max_{1\le i\le ne} \frac{e_i}{f(\boldsymbol{x}_i)}; \quad MEAE_{rel} = \frac{1}{ne}\sum_{i=1}^{ne} \frac{e_i}{f(\boldsymbol{x}_i)}; \quad RMSE_{rel} = \sqrt{\frac{1}{ne}\sum_{i=1}^{ne}\left(\frac{e_i}{f(\boldsymbol{x}_i)},\right)^2}$$

by varying the degree r from 1 to 6.

In Table 4 we report the number s of m-tuples by varying the degree r of the local polynomial interpolant and the CPU time (in seconds) to reconstruct the surface in Figure 2 by using 412×164 grid points in the rectangle $[0,2060]\times[0,818]$ as evaluation points.

In Figure 2 we display the surface obtained by using the 95020 mock-Halton points and by evaluating the multinode Shepard operator on 412×164 grid points in the rectangle $[0,2060]\times[0,818]$.

The numerical results clearly show the convenience of the use of Algorithm 2 instead of Algorithm 1 since it produces approximations of the same accuracies of Algorithm 1 with

	number of m-tuples		CPU time (sec)	
r	Alg. 1	Alg. 2	Alg. 1	Alg. 2
1	94065	56592	808	262
2	95020	34826	1350	236
3	95018	25733	1644	241
4	95019	20200	2313	265
5	95019	16248	2891	271
6	95018	13339	3679	281

Table 4: Number of m-tuples varying the degree r of the local polynomial interpolant and CPU time in seconds to reconstruct the surface in Figure 2 by using, as evaluation points, 412×164 grid points in the rectangle $[0,2060]\times[0,818]$.

significantly reduced computation cost and CPU time. More precisely, Table 4 shows that Algorithm 2 reduces the number s of subsets σ_j as the degree r increases of a factor approximately equal to $\frac{n}{r+1}$, where n is the number of interpolation nodes.

3 Conclusions

We encourage scientists involved in image reconstruction to explore our MATLAB package `Multinode_Shepard` for enhanced accuracy and efficiency in scattered data interpolation. This package implements the multinode Shepard method, a significant advancement over the classical Shepard's method, providing robust solutions for interpolating complex datasets, with reduced computational cost and improved accuracy.

The multinode Shepard method, as demonstrated in our numerical experiments, significantly improves interpolation accuracy by utilizing local polynomial interpolants and inverse distance weighting functions. This method has been rigorously tested on real-world data, such as the Stromboli Volcano dataset, showing remarkable efficiency and precision.

By using our MATLAB package, you can benefit from:

- high accuracy in data interpolation, even with irregularly distributed datasets;
- efficient computation through optimized algorithms, reducing processing time and computational costs;
- robust performance validated through extensive numerical experiments.

The `Multinode_Shepard` package is freely available, and we invite you to access the source code and demonstrations via our GitHub repository: `Multinode_Shepard` GitHub [12]. This tool is designed to assist in your research, enabling more precise and reliable image reconstructions.

We welcome feedback and suggestions from the research community and are open to collaborations that can further enhance the capabilities and applications of the multinode Shepard method. Please feel free to reach out to us for any inquiries or potential joint projects.

Acknowledgments

This research has been achieved as part of RITA "Research ITalian network on Approximation", as part of the UMI group "Teoria dell'Approssimazione e Applicazioni" and was supported by INDAM-GNCS project 2024. The authors are members of the INdAM Research group GNCS.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

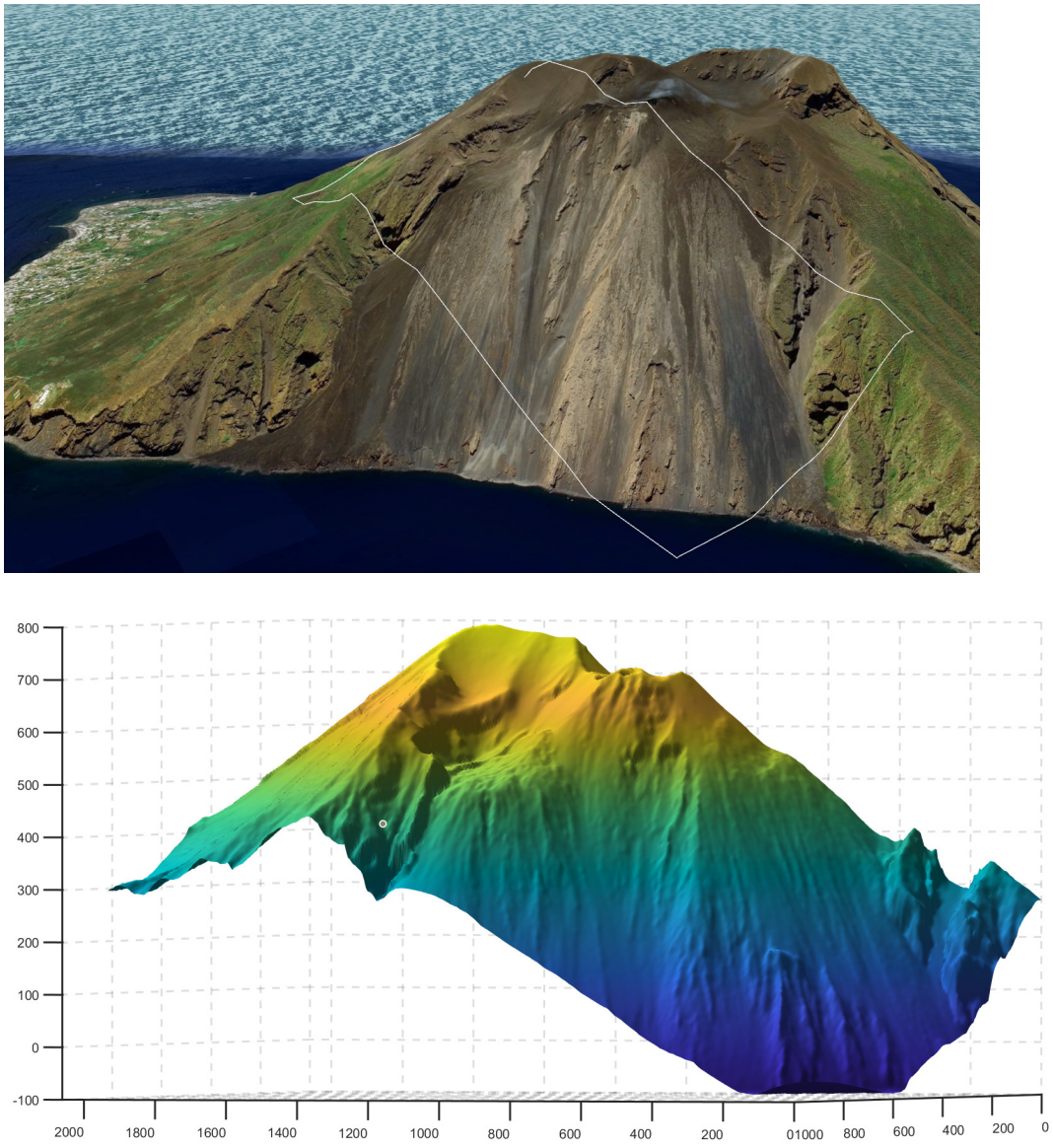


Figure 2: The Google Heart image of the Stromboli Volcano with the evidence of the zone from which the DEM data come from (top) and the reconstructed surface of the Stromboli Volcano (bottom) evaluated at a regular grid of 412×164 points in the rectangle $[0,2060] \times [0,818]$.

References

[1] L. Bos, S. De Marchi, A. Sommariva, and M. Vianello. Computing multivariate Fekete and Leja points by numerical linear algebra. *SIAM J. Numer. Anal.*, 48(5):1984–1999, 2010. ISSN 0036-1429,1095-7170. doi: 10.1137/090779024. URL https://doi.org/10.1137/090779024.

[2] B. Carr. Stromboli Volcano (Vents & Sciara del Fuoco): Italy, September 10, 2018. https://doi.org/10.5069/G9VX0DNG, 2019. Accessed: 2024-07-09.

[3] R. Cavoretto, A. De Rossi, F. Dell'Accio, and F. Di Tommaso. An efficient trivariate algorithm for tetrahedral Shepard interpolation. *J. Sci. Comput.*, 82(3):Paper No. 57, 15, 2020. ISSN 0885-7474,1573-7691. doi: 10.1007/s10915-020-01159-3. URL https://doi.org/10.1007/s10915-020-01159-3.

[4] F. Dell'Accio and F. Di Tommaso. Scattered data interpolation by Shepard's like methods: classical results and recent advances. *Dolomites Res. Notes Approx.*, 9(2):32–44, 2016. ISSN 2035-6803. doi: 10.14658/PUPJ-DRNA-2016-Special_Issue-5. URL https://drna.padovauniversitypress.it/2016/Special_Issue/5.

[5] F. Dell'Accio and F. Di Tommaso. Rate of convergence of multinode Shepard operators. *Dolomites Res. Notes Approx.*, 12(1):1–6, 2019. ISSN 2035-6803. doi: 10.1016/j.physletb.2018.10.073. URL https://doi.org/10.1016/j.physletb.2018.10.073.

[6] F. Dell'Accio and F. Di Tommaso. On the hexagonal Shepard method. *Appl. Numer. Math.*, 150:51–64, 2020. ISSN 0168-9274,1873-5460. doi: 10.1016/j.apnum.2019.09.005. URL https://doi.org/10.1016/j.apnum.2019.09.005.

[7] F. Dell'Accio, F. Di Tommaso, and K. Hormann. On the approximation order of triangular Shepard interpolation. *IMA J. Numer. Anal.*, 36(1):359–379, 2016. ISSN 0272-4979,1464-3642. doi: 10.1093/imanum/dru065. URL https://doi.org/10.1093/imanum/dru065.

[8] F. Dell'Accio, F. Di Tommaso, O. Nouisser, and N. Siar. Solving Poisson equation with Dirichlet conditions through multinode Shepard operators. *Comput. Math. Appl.*, 98:254–260, 2021. ISSN 0898-1221,1873-7668. doi: 10.1016/j.camwa.2021.07.021. URL https://doi.org/10.1016/j.camwa.2021.07.021.

[9] F. Dell'Accio, F. Di Tommaso, and N. Siar. On the numerical computation of bivariate Lagrange polynomials. *Appl. Math. Lett.*, 112:Paper No. 106845, 9, 2021. ISSN 0893-9659,1873-5452. doi: 10.1016/j.aml.2020.106845. URL https://doi.org/10.1016/j.aml.2020.106845.

[10] F. Dell'Accio, F. Di Tommaso, N. Siar, and M. Vianello. Numerical differentiation on scattered data through multivariate polynomial interpolation. *BIT*, 62(3):773–801, 2022. ISSN 0006-3835,1572-9125. doi: 10.1007/s10543-021-00897-6. URL https://doi.org/10.1007/s10543-021-00897-6.

[11] F. Dell'Accio, A. Sommariva, and M. Vianello. Random sampling and unisolvent interpolation by almost everywhere analytic functions. *Appl. Math. Lett.*, 145:Paper No. 108734, 6, 2023. ISSN 0893-9659,1873-5452. doi: 10.1016/j.aml.2023.108734. URL https://doi.org/10.1016/j.aml.2023.108734.

[12] F. Dell'Accio, F. Di Tommaso, and F. Larosa. Multinode Shepard method: the MATLAB Implementation. https://github.com/FilomenaDiTommaso85/Multinode_Shepard.git, 2024.

[13] R. Farwig. Rate of convergence of Shepard's global interpolation formula. *Math. Comp.*, 46(174):577–590, 1986. ISSN 0025-5718,1088-6842. doi: 10.2307/2007995. URL https://doi.org/10.2307/2007995.

[14] F. F. Little. Convex combination surfaces. In R. E. Barnhill and W. Boehm, editors, *Surfaces in computer aided geometric design - Proceedings of a Conference held at Oberwolfach, April 25–30, 1982*, volume 1479, pages 99–108. North-Holland Publishing Co., Amsterdam-New York, 1983.

[15] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *ACM '68: Proceedings of the 1968 23rd ACM National Conference*, pages 517–524, New York, NY, United States, 1968. Association for Computing Machinery. doi: 10.1145/800186.810616.