Dolomites Research Notes on Approximation

Special Issue *ATMA2021 - Approximation: Theory, Methods, and Applications*, Volume 15 · 2022 · Pages 56–64

Computational issues by interpolating with inverse multiquadrics: a solution

Stefano De Marchi[a] · Nadaniela Egidi[b] · Josephin Giacomini[b] · Pierluigi Maponi[b] · Alessia Perticarini[b]

Abstract

We consider the interpolation problem with the inverse multiquadric radial basis function. The problem usually produces a large dense linear system that has to be solved by iterative methods. The efficiency of such methods is strictly related to the computational cost of the multiplication between the coefficient matrix and the vectors computed by the solver at each iteration. We propose an efficient technique for the calculation of the product of the coefficient matrix and a generic vector. This computation is mainly based on the well-known spectral decomposition in spherical coordinates of the Green's function of the Laplacian operator. We also show the efficiency of the proposed method through numerical simulations.

1 Introduction

Radial basis functions (RBFs) are efficient tools in approximation of functions and data, in the solution of many engineering problems [16], including applications in image processing [11, 7]. They are a suitable tool for scattered data interpolation problems [17], solution of differential equations [12], machine learning techniques [21] and other applications [16]. In particular, the solution of scattered data interpolation problems leads to a linear system that has a dense matrix of coefficients. We can identify three main difficulties in solving the corresponding linear system: (i) high computational cost when a large number of interpolation points is considered, (ii) numerical instability and (iii) a clever choice of the shape parameter.

In the present paper, we deal with the computational cost of the solution of the linear systems in the case of large interpolation problems. For these linear systems, iterative methods are the standard solvers and the efficiency of these methods can be improved in several ways, for instance, by improving the efficiency of the iterate and/or by providing an effective preconditioner [1] or decreasing the computational cost of each iteration.

The *fast multipole method* [14] has been proposed to reduce the computational cost in the solution algorithms for integral equations arising in classical scattering theory. This gave rise to a set of effective techniques centred around the principle of *divide et impera*. Among the many existing variants of this idea, several schemes have been developed to provide efficient solution strategies for structured linear systems [24, 25], eigenvalue problems [5, 23], interpolation problems by RBFs [9, 2, 3, 4], integral equations [10, 19, 13], and partial differential equations [18]. A review of similar techniques can be found in [6].

In this work, we propose a method aimed to decrease the computational cost of the product of the coefficient matrix and a generic vector. The method is based on a local low-rank representation of the interpolation matrix A. This representation resembles ones used in the fast multipole method and is conceptually based on a simple remark: the computational cost of the product of a matrix A of order N and a generic vector is proportional to N^2, but if a matrix A admits a decomposition of the form $A = UV$, where U and V are rectangular matrices $N \times p$ and $p \times N$, respectively, the action of A has a computational cost proportional to $2Np$, so, this decomposition can be profitably used when $p \ll N$. The efficiency of the proposed technique is also shown through numerical experiments.

The paper is organised as follows. In Section 2, we describe the proposed method, in particular, we define the interpolation matrix decomposition, which is based on the spectral expansion of the fundamental solution of the Laplacian operator, and we introduce the translation strategy for the efficient computation of the matrix action. In Section 3, we show the results of some numerical experiments. Finally, we give some conclusions and future developments in Section 4.

2 The interpolation problem with inverse multiquadric RBFs

We consider the interpolation problem where the interpolation function is expressed in terms of inverse multiquadric RBFs. In the following, we focus on interpolation problem on $\mathbb{R}^2$ but the results can be easily generalized in $\mathbb{R}^s$, for different s. Initially, we summarise some preliminary concepts and fix notations. Let Ω be a subset of $\mathbb{R}^2$, $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \subset \Omega$ be the set of N distinct points, usually called *data sites*, and $f_i \in \mathbb{R}$ be the *data values*. Moreover, data values are supposed to be obtained from some unknown function $f : \Omega \to \mathbb{R}$ evaluated at the data sites, i.e.

$$f(\boldsymbol{x}_i) = f_i, \qquad i = 1, 2, \ldots, N. \tag{1}$$

[a] Department of Mathematics "Tullio Levi-Civita", University of Padua, Italy
[b] School of Science and Technology - Mathematics Division, University of Camerino, via Madonna delle Carceri 9, 62032 Camerino, Italy

Given a RBF defined through $\phi : [0,\infty) \to \mathbb{R}$, $\phi(r_j) = \phi(\|\mathbf{x}-\mathbf{y}_j\|)$ for $r_j = \|\boldsymbol{x}-\boldsymbol{y}_j\|$, the interpolation problem consists in finding the approximation function $P_f(\boldsymbol{x})$ as follows

$$P_f(\boldsymbol{x}) = \sum_{j=1}^{N} c_j \phi(\|\boldsymbol{x}-\boldsymbol{y}_j\|), \qquad \boldsymbol{x} \in \Omega, \tag{2}$$

where $\|\cdot\|$ is the Euclidean norm, $\boldsymbol{y}_j$, $j = 1,\dots,N$, are the centres of the RBFs and coincide with the data sites $\boldsymbol{y}_j = \boldsymbol{x}_j$, and c_j, $j = 1,\dots,N$, are unknown coefficients. We want to determine these unknowns in such a way that:

$$P_f(\boldsymbol{x}_i) = f_i, \qquad i = 1,2,\dots,N. \tag{3}$$

In order to compute P_f in (2), we have to solve the system of linear equations (3) which has the form:

$$A\mathbf{c} = \mathbf{f}, \tag{4}$$

where the interpolation matrix A has entries $a_{ij} = \phi(\|\boldsymbol{x}_i-\boldsymbol{y}_j\|)$, $i,j = 1,\dots,N$, $\mathbf{c} = (c_1,\dots,c_N)^T$ is the unknown coefficient vector, and $\mathbf{f} = (f_1,\dots,f_N)^T$ is the known term given by the interpolation condition (3).

The linear system (4) has a dense coefficients matrix, moreover, for large values of N direct methods usually cannot be used to calculate its numerical solution due to the memory and CPU-time resources required. Thus, iterative methods remain the unique numerical alternative for dealing with such kind of linear systems. In general, the efficiency of such methods depends on the spectral and sparsity properties of the coefficient matrix. The former usually influence the number of iterations required to achieve a given accuracy in the numerical solution, while the latter influence the computational cost of each iteration, which is strictly dependent on the iterative method considered but, for large dense linear systems, most of this cost is due to the computation of the product of the corresponding coefficient matrix and the generic tentative solution. We point out that, from standard arguments on numerical linear algebra, the computational cost of this matrix-vector product is proportional to N^2, when N is the order of the matrix, so we need to supply a more efficient technique for this computation.

For the description of the proposed method, we consider the interpolation problem in $\mathbb{R}^2$ with inverse multiquadric (IMQ-) RBFs, even if, this study can be extended to other families of RBFs and in other dimensions. More precisely, let $\boldsymbol{y} \in \mathbb{R}^2$ be the center of the IMQ-RBFs, $\boldsymbol{x} \in \mathbb{R}^2$, and let $t \in \mathbb{R}$ be the shape parameter, we consider:

$$\phi(\|\boldsymbol{x}-\boldsymbol{y}\|) = \frac{1}{\sqrt{t^2+\|\boldsymbol{x}-\boldsymbol{y}\|^2}}. \tag{5}$$

In the following sections, we introduce the proposed technique. In particular, in Section 2.1, we present the spectral expansion in spherical coordinates of the Green's function of the Laplacian operator and its use in the local decomposition of the matrix A. In Section 2.2, we discuss the translation strategy for the efficient computation of the product of the matrix A and a generic vector. Then, in Section 2.3, we analyse the computational cost of the proposed technique.

2.1 The Green's function of the Laplacian operator

The Green's function, $G(\boldsymbol{x};\boldsymbol{y})$, of a linear differential operator L is a solution of the equation $LG(\boldsymbol{x};\boldsymbol{y}) = \delta(\boldsymbol{x}-\boldsymbol{y})$, where δ is the Dirac delta. From standard arguments, we have that

$$G(\boldsymbol{X};\boldsymbol{Y}) = \frac{1}{\|\boldsymbol{X}-\boldsymbol{Y}\|}, \qquad \boldsymbol{X},\boldsymbol{Y} \in \mathbb{R}^3, \boldsymbol{X} \neq \boldsymbol{Y}, \tag{6}$$

is the Green's function of the Laplacian operator $L = \Delta$ [8], moreover, $G(\boldsymbol{X};\boldsymbol{Y})$ has a well-known spectral decomposition in spherical coordinates. In particular if $(\rho_x,\theta_x,\omega_x)$, $(\rho_y,\theta_y,\omega_y)$ denote the spherical coordinates of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively, where $\rho_x,\rho_y \in [0,+\infty)$, $\theta_x,\theta_y \in [0,\pi]$, $\omega_x,\omega_y \in [0,2\pi)$, we have the following spectral expansion of $G(\boldsymbol{X};\boldsymbol{Y})$ when $\rho_x \neq \rho_y$

$$\begin{aligned} G(\boldsymbol{X};\boldsymbol{Y}) = \sum_{n=0}^{\infty}\sum_{m=0}^{n} \epsilon_m \frac{(n-m)!}{(n+m)!} P_n^m\left(\cos\theta_y\right) P_n^m\left(\cos\theta_x\right) \cdot \\ \cdot \cos\left(m(\omega_x-\omega_y)\right) \begin{cases} \rho_x^n/\rho_y^{n+1}, & \text{if } \rho_y > \rho_x, \\ \rho_y^n/\rho_x^{n+1}, & \text{if } \rho_x > \rho_y, \end{cases} \end{aligned} \tag{7}$$

where ϵ_m is the Neumann factor, that is $\epsilon_0 = 1, \epsilon_m = 2$, $m > 0$, and P_n^m is the associated Legendre function of order m and degree n, see [20] for more details. Furthermore, in a practical application of (7), we have to consider the truncated series, that is

$$\begin{aligned} G(\boldsymbol{X};\boldsymbol{Y}) \approx \sum_{n=0}^{M}\sum_{m=0}^{n} \epsilon_m \frac{(n-m)!}{(n+m)!} P_n^m\left(\cos\theta_y\right) P_n^m\left(\cos\theta_x\right) \cdot \\ \cdot \cos\left(m(\omega_x-\omega_y)\right) \begin{cases} \rho_x^n/\rho_y^{n+1}, & \text{if } \rho_y > \rho_x, \\ \rho_y^n/\rho_x^{n+1}, & \text{if } \rho_x > \rho_y, \end{cases} \end{aligned} \tag{8}$$

where $M \in \mathbb{N}$ is the truncation parameter for the index n in (7). The accuracy of approximation (8) strongly depends on the particular choice of $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^3$. In particular, it depends on the ratio between ρ_y and ρ_x, as stated in the following theorem.

Theorem 2.1. *Let $\boldsymbol{X},\boldsymbol{Y}\in\mathbb{R}^3$ be such that $\rho_y<\rho_x$ and let $r=\dfrac{\rho_y}{\rho_x}$. For the absolute error E in the truncated series (8), that is*

$$E(\boldsymbol{X},\boldsymbol{Y})=\left|G(\boldsymbol{X};\boldsymbol{Y})-\sum_{n=0}^{M}\sum_{m=0}^{n}\epsilon_m\frac{(n-m)!}{(n+m)!}P_n^m\left(\cos\theta_y\right)P_n^m\left(\cos\theta_x\right)\cos\left(m(\omega_x-\omega_y)\right)\frac{1}{\rho_x}r^n\right|,$$

we have the following bound

$$E(\boldsymbol{X},\boldsymbol{Y})\le\frac{1}{\rho_x}\frac{r^{M+1}}{(1-r)}. \tag{9}$$

Proof. Let α be the angle between $\boldsymbol{X}$ and $\boldsymbol{Y}$, we have:

$$\begin{aligned}E(\boldsymbol{X},\boldsymbol{Y})&=\left|\sum_{n=M+1}^{\infty}\sum_{m=0}^{n}\epsilon_m\frac{(n-m)!}{(n+m)!}P_n^m\left(\cos\theta_y\right)P_n^m\left(\cos\theta_x\right)\cos\left(m(\omega_x-\omega_y)\right)\frac{1}{\rho_x}r^n\right|\\&=\left|\frac{1}{\rho_x}\sum_{n=M+1}^{\infty}P_n(\cos\alpha)\,r^n\right|\le\frac{1}{\rho_x}\sum_{n=M+1}^{\infty}\left|P_n(\cos\alpha)\right|r^n\le\frac{1}{\rho_x}\sum_{n=M+1}^{\infty}r^n=\\&=\frac{1}{\rho_x}\left(\sum_{n=0}^{\infty}r^n-\sum_{n=0}^{M}r^n\right)=\frac{1}{\rho_x}\left(\frac{1}{1-r}-\frac{1-r^{M+1}}{1-r}\right)=\frac{1}{\rho_x}\frac{r^{M+1}}{1-r}\end{aligned} \tag{10}$$

where P_n is the Legendre polynomial of degree n and the second equality holds because

$$P_n(\cos\alpha)=\sum_{m=0}^{n}\epsilon_m\frac{(n-m)!}{(n+m)!}P_n^m\left(\cos\theta_y\right)P_n^m\left(\cos\theta_x\right)\cos\left(m(\omega_x-\omega_y)\right),$$

see [20, Formula (10.3.38)] for more details, and the last inequality holds because $|P_n(x)|\le 1$ for all $|x|\le 1$ [22, Formula (28.35)]. □

We note that a similar theorem holds in the case $\rho_x<\rho_y$ and $r=\dfrac{\rho_x}{\rho_y}$. As a consequence of Theorem 2.1, we have that the series in (7) converges uniformly on every compact set of the domain $||\boldsymbol{X}||>||\boldsymbol{Y}||$ or on the domain $||\boldsymbol{X}||<||\boldsymbol{Y}||$. In Figure 1, we report the result of a numerical experiment showing that, when $\boldsymbol{X}$ is sufficiently far from $\boldsymbol{Y}$, formula (8) is accurate even with moderate values of M, while it produces large errors when $||\boldsymbol{X}||\approx||\boldsymbol{Y}||$, even if a large truncation index M is used. More precisely, in Figure 1, $\rho_x\in\left[\dfrac{11}{10},21\right]$ and $\rho_y=1$; in addition, $\theta_x=\theta_y=\omega_x=\omega_y=\dfrac{\pi}{3}$ for the black curves in Figure 1, whereas $\theta_x=\pi$, $\theta_y=\dfrac{\pi}{4}$, $\omega_x=\dfrac{\pi}{3}$ and $\omega_y=\dfrac{\pi}{2}$ for blue curves, in Figure 1.

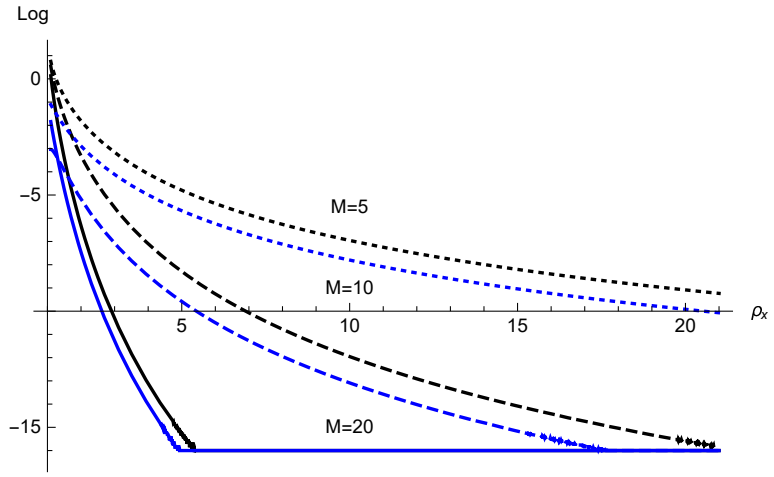


Figure 1: The error $E(\boldsymbol{X},\boldsymbol{Y})$ in the representation formula (8) with truncation index $M=5$, $M=10$, $M=20$, and $\rho_y=1$, when $\theta_x=\theta_y=\omega_x=\omega_y=\dfrac{\pi}{3}$ in black, and $\theta_x=\pi$, $\theta_y=\dfrac{\pi}{4}$, $\omega_x=\dfrac{\pi}{3}$, $\omega_y=\dfrac{\pi}{2}$ in blue. We note that the logarithmic scale is used for the y-axis.

2.2 The translation technique

If we consider $\boldsymbol{X}=(x_1,x_2,t/2)$, $\boldsymbol{Y}=(y_1,y_2,-t/2)$, where t is the shape parameter in (5), $\boldsymbol{x}=(x_1,x_2)\in\mathbb{R}^2$, and $\boldsymbol{y}=(y_1,y_2)\in\mathbb{R}^2$, we have that

$$G(\boldsymbol{X};\boldsymbol{Y})=\frac{1}{||\boldsymbol{X}-\boldsymbol{Y}||}=\frac{1}{\sqrt{||\boldsymbol{x}-\boldsymbol{y}||^2+t^2}}=\phi(||\boldsymbol{x}-\boldsymbol{y}||) \tag{11}$$

and Formula (7) gives the spectral decomposition for the IMQ-RBFs with shape parameter t. We note that Formula (7) resembles the usual representation of degenerate kernels, in fact, it expresses $\phi(||\boldsymbol{x}-\boldsymbol{y}||)$ as a series, where each term is a product of two functions of only one variable, i.e., a function of $\boldsymbol{x}$ and the other of $\boldsymbol{y}$. In addition, it identifies two regions where we have two different representations. In fact, the structure of (7) requires us to divide the elements of $\mathbf{A}$ into three sets: the elements where $||\boldsymbol{x}|| < ||\boldsymbol{y}||$, those where $||\boldsymbol{x}|| > ||\boldsymbol{y}||$, and those where $||\boldsymbol{x}|| = ||\boldsymbol{y}||$ for which (7) cannot be used. This is an interesting formula resembling a low-rank decomposition of matrix A, but unfortunately it does not provide a low-rank representation of the matrix A due to the aforementioned partition of A. However, formula (7) provides a local decomposition for $\phi(||\boldsymbol{x}-\boldsymbol{y}||)$ that can be profitably used in the solution of (4).

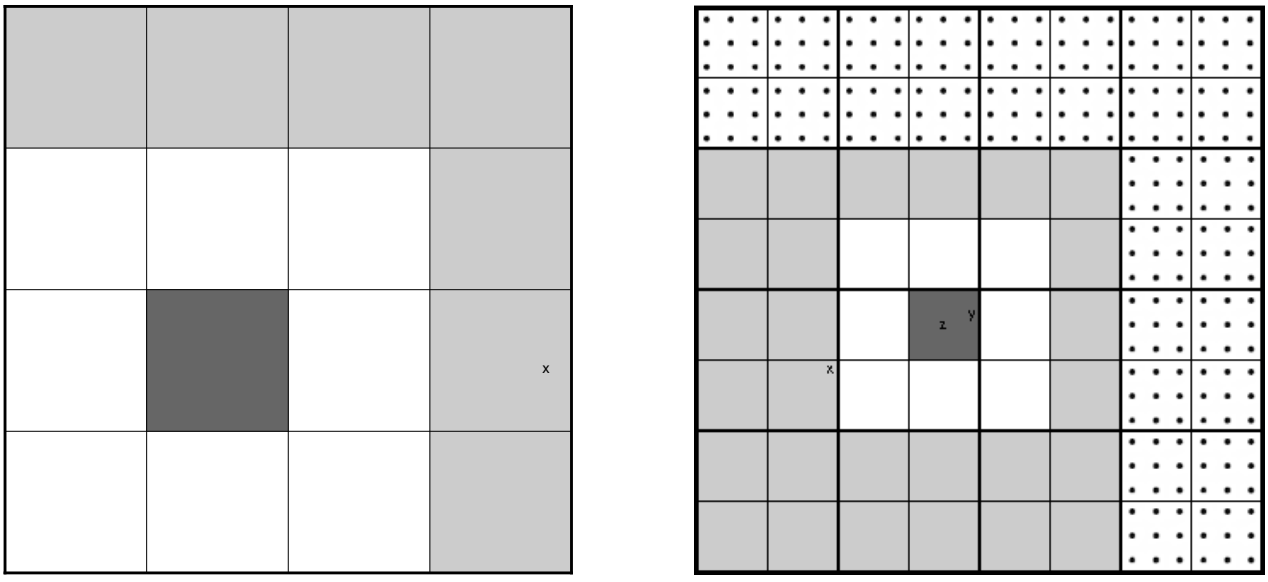
When the ratio $\rho_y/\rho_x \approx 1$ (but $\boldsymbol{X}$ is substantially different from $\boldsymbol{Y}$), a simple translation operation can be used in formula (8). Let $\boldsymbol{z} = (z_1, z_2) \in \Omega \subset \mathbb{R}^2$ be such that $||\boldsymbol{y}-\boldsymbol{z}|| < ||\boldsymbol{x}-\boldsymbol{z}||$, and let $\boldsymbol{Z} = (z_1, z_2, -t/2)$, then from (8) we have:

$$\begin{aligned}\phi(||\boldsymbol{x}-\boldsymbol{y}||) &= \phi(||(\boldsymbol{x}-\boldsymbol{z})-(\boldsymbol{y}-\boldsymbol{z})||) = G(\boldsymbol{X}-\boldsymbol{Z};\boldsymbol{Y}-\boldsymbol{Z}) \approx \\ &\approx \sum_{n=0}^{M}\sum_{m=0}^{n} \epsilon_m \frac{(n-m)!}{(n+m)!} P_n^m\left(\cos(\theta_{y-z})\right) P_n^m\left(\cos(\theta_{x-z})\right) \cos\left(m(\omega_{x-z}-\omega_{y-z})\right) \frac{\rho_{y-z}^n}{\rho_{x-z}^{n+1}},\end{aligned} \tag{12}$$

where $(\rho_{x-z}, \theta_{x-z}, \omega_{x-z})$, $(\rho_{y-z}, \theta_{y-z}, \omega_{y-z})$ are the spherical coordinates of vectors $\boldsymbol{X}-\boldsymbol{Z}$, $\boldsymbol{Y}-\boldsymbol{Z} \in \mathbb{R}^3$, respectively. The translation technique for the computation of $A\mathbf{c}$ consists in a convenient use of (12). For simplicity of notation, let us define the following quantities: $d_{n,m} = \epsilon_m \dfrac{(n-m)!}{(n+m)!}$, $h_{n,m}(\boldsymbol{X}-\boldsymbol{Z}) = \dfrac{P_n^m(\cos(\theta_{x-z}))}{\rho_{x-z}^{n+1}}$, $j_{n,m}(\boldsymbol{Y}-\boldsymbol{Z}) = P_n^m\left(\cos(\theta_{y-z})\right)\rho_{y-z}^n$. Thus, when $||\boldsymbol{y}-\boldsymbol{z}|| < ||\boldsymbol{x}-\boldsymbol{z}||$, we can separate the contribution of $\boldsymbol{x}$ and $\boldsymbol{y}$ in the following way:

$$\begin{aligned}\phi(||\boldsymbol{x}-\boldsymbol{y}||) \approx &\sum_{n=0}^{M}\sum_{m=0}^{n} d_{n,m} h_{n,m}(\boldsymbol{X}-\boldsymbol{Z}) \cos(m\omega_{x-z}) j_{n,m}(\boldsymbol{Y}-\boldsymbol{Z}) \cos(m\omega_{y-z}) + \\ &+ \sum_{n=0}^{M}\sum_{m=0}^{n} d_{n,m} h_{n,m}(\boldsymbol{X}-\boldsymbol{Z}) \sin(m\omega_{x-z}) j_{n,m}(\boldsymbol{Y}-\boldsymbol{Z}) \sin(m\omega_{y-z}).\end{aligned} \tag{13}$$

We can suppose, without losing generality, that the domain Ω is contained in a square $D \subset \mathbb{R}^2$. In fact, all the approximation formulas in D are formally valid also in $\Omega \subset D$, however, the analysis of this method (like the computational cost) can depend on the particular shape of the domain Ω. The proposed strategy considers a set of partitions of D: the first partition is obtained by dividing D into 4×4 equivalent blocks, the successive are obtained by bisecting the edges of the previous blocks. More precisely, the partition considered in D depends on a recursive index l, with $l = 1, 2, \ldots, L$. For each level l, the square D is partitioned in $2^{l+1} \times 2^{l+1}$ blocks like in Figure 2, where the first two levels of these partitions are considered. The idea of the proposed strategy is the following: when $\boldsymbol{y} \in \mathcal{X}$ is in a set S (the dark gray squares in Figure 2) of the considered partition, we associate to $\boldsymbol{y}$ the center $\boldsymbol{z}$ of S and we use formula (12) only for $\boldsymbol{x} \in \mathcal{X}$ belonging to well-separated squares of the considered partition (the light gray squares in Figure 2). In this way, we obtain a high accuracy of formula (12), even for small values of the truncation index M.



(a) **(b)**

Figure 2: Example of the translation technique at levels $l = 1$ (a) and $l = 2$ (b).

In more detail, for each point $\boldsymbol{y}$ in a block at level 1, we choose $\boldsymbol{z}$ as the center of this block as shown in the dark gray block in Figure 2(a). Then, we define the well-separated blocks, with respect to the previously selected block that contains $\boldsymbol{y}$, as those

blocks of points $\boldsymbol{x}$ such that $\frac{\rho_{y-z}}{\rho_{x-z}} < \frac{1}{2}$. In Figure 2(a), the well-separated blocks are the light gray ones. So, for each $\boldsymbol{y}$, in a given block of the partition we apply formula (12) for points $\boldsymbol{x}$ in the corresponding well-separated blocks of the same partition. In this way, if $t = 1$ and D has unitary edge, we have an error $E < 2.86 \cdot 10^{-9}$ when $M = 10$, and an error $E < 4.41 \cdot 10^{-17}$ when $M = 20$. In fact, it is easy to verify that $\rho_{y-z} \leq \frac{\sqrt{2}}{8}$ and $\rho_{x-z} \geq \frac{\sqrt{73}}{8}$, so that $\frac{\rho_{y-z}}{\rho_{x-z}} \leq 0.1655$. At the second level, i.e. $l = 2$, each block of level 1 is partitioned in 2×2 equivalent squares, thus obtaining 64 blocks, as in Figure 2(b). At level 2, the same procedure is applied to the part of the domain D that has not already been considered at level 1. So, for each point $\boldsymbol{y}$ in a smaller block, as the dark gray block in Figure 2(b), we choose $\boldsymbol{z}$ as the center of the block, and we apply formula (12) for each point $\boldsymbol{x}$ in the new well-separated blocks, i.e., the light gray blocks in Figure 2(b). Also at this level we have the same accuracy of previous level. The dotted region in Figure 2(b) highlights the region where formula (12) has been already applied and so it does not need to be treated at this level. We repeat this procedure for all the blocks until the finest level L is reached, where the contributions of the remaining white regions are directly computed by using formula (5) in the matrix action.

In Algorithm 1, we report the translation technique for computing the matrix-vector product $\mathbf{b} = A\mathbf{u}$, when $\mathbf{u} \in \mathbb{R}^N$ is a generic vector. In this algorithm, we use the following notation: P_l is the set of all blocks at level l, p is one of these blocks, S_p is the set of the blocks well-separated from the block p, and NS_p is the set of the blocks which are not well-separated from the block p.

Algorithm 1: Given $\mathbf{u} \in \mathbb{R}^N$, computes $\mathbf{b} = A\mathbf{u} \in \mathbb{R}^N$ as follows.

$\mathbf{b} = (0, \ldots, 0)^T$;
for $l = 1, \ldots, L$ **do**
  for $p \in P_l$ **do**
    $\boldsymbol{z}$ is the center of p
    for $\boldsymbol{y}_j \in p$ **do**
      $\mathbf{v} = (0, \ldots, 0)^T$;
      $\mathbf{w} = (0, \ldots, 0)^T$;
      Compute the contribution of $\boldsymbol{y}_j$ from formula (13)
      for $n = 0, \ldots, M$ **do**
        for $m = 0, \ldots, n$ **do**
          $k = \frac{n(n+1)}{2} + m + 1$
          $\mathbf{v}_k = \mathbf{v}_k + j_{n,m}(\boldsymbol{Y}_j - \boldsymbol{Z}) \sin(m\omega_{y_j - z})\mathbf{u}_j$
          $\mathbf{w}_k = \mathbf{w}_k + j_{n,m}(\boldsymbol{Y}_j - \boldsymbol{Z}) \cos(m\omega_{y_j - z})\mathbf{u}_j$
        end
      end
    end
    for $\boldsymbol{x}_i \in S_p$ **do**
      Compute the contribution of $\boldsymbol{x}_i$ from formula (13)
      for $n = 0, \ldots, M$ **do**
        for $m = 0, \ldots, n$ **do**
          $k = \frac{n(n+1)}{2} + m + 1$
          $\mathbf{b}_i = \mathbf{b}_i + d_{n,m} h_{n,m} (\boldsymbol{X}_i - \boldsymbol{Z}) \big[\sin(m\omega_{x_i - z})\mathbf{v}_k + \cos(m\omega_{x_i - z})\mathbf{w}_k \big]$
        end
      end
    end
  end
end
for $p \in P_L$ **do**
  for $\boldsymbol{x}_i \in NS_p$ **do**
    for $\boldsymbol{y}_j \in p$ **do**
      Compute the contribution of $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$ from formula (4)
      $\mathbf{b}_i = \mathbf{b}_i + A_{i,j}\mathbf{u}_j$
    end
  end
end

2.3 The computational cost

The recursive procedure described in the previous section gives an efficient method for computing $A\mathbf{u}$, where $\mathbf{u} \in \mathbb{R}^N$ is a generic vector and the data sites in $\mathcal{X}$ are generic points of the domain Ω. However, for the sake of simplicity, we suppose a uniform

distribution of points, so at the generic level l, each block p contains $\frac{N}{4^{l+1}}$ data sites. As illustrated in Algorithm 1, we have that

$$\sum_{j=1}^{N} A_{i,j}\mathbf{u}_j = \sum_{l=1}^{L}\sum_{\substack{p\in P_l\\ i\in S_p}}\left\{\sum_{n=0}^{M}\sum_{m=0}^{n}\left(d_{n,m}h_{n,m}(\boldsymbol{X}_i-\boldsymbol{Z})\cos(m\omega_{x_i-z})\right)\left(\sum_{j\in p} j_{n,m}(\boldsymbol{Y}_j-\boldsymbol{Z})\cos(m\omega_{y_j-z})\mathbf{u}_j\right)+\right.$$
$$\left.+\sum_{n=0}^{M}\sum_{m=0}^{n}\left(d_{n,m}h_{n,m}(\boldsymbol{X}_i-\boldsymbol{Z})\sin(m\omega_{x_i-z})\right)\left(\sum_{j\in p} j_{n,m}(\boldsymbol{Y}_j-\boldsymbol{Z})\sin(m\omega_{y_j-z})\mathbf{u}_j\right)\right\}+ \tag{14}$$
$$+\sum_{\substack{p\in P_L\\ i\in NS_p}}\left\{\sum_{j\in p} A_{i,j}\mathbf{u}_j\right\}, \qquad i=1,\dots,N,$$

where $i\in S_p$ denotes that $\boldsymbol{x}_i$ is in well-separated blocks S_p, while $j\in p$ denotes that $\boldsymbol{y}_j$ is in p. This shows that the computation consists of two main addenda. The first addendum in (14), i.e., the one for $i\in S_p$, $p\in P_l$, $l=1,\dots,L$, can be computed efficiently since the part depending on the row indices i is independent of the part depending on the column indices j; while the second addendum has to be evaluated only at the last level L. The computational cost c of the matrix-vector product in (14) can be obtained by summing the cost of the various steps. We retrace the computations in Algorithm 1 and we evaluate the overall computational cost by counting only the multiplication operations. The computation of the coefficients $d_{n,m}$ and functions $h_{n,m}, j_{n,m}$ as well as trigonometric functions in formula (14) are not considered since they are computed only once at the beginning of the solution process and stored in arrays.

Let c_1 be the cost of lines $5-16$ in Algorithm 1, so c_1 is equal to the cost of

$$\mathbf{v}_{n,m}(\boldsymbol{Z})=\sum_{j\in p} j_{n,m}(\boldsymbol{Y}_j-\boldsymbol{Z})\sin(m\omega_{y_j-z})\mathbf{u}_j \quad \text{and} \quad \mathbf{w}_{n,m}(\boldsymbol{Z})=\sum_{j\in p} j_{n,m}(\boldsymbol{Y}_j-\boldsymbol{Z})\cos(m\omega_{y_j-z})\mathbf{u}_j, \tag{15}$$

for all $n=0,1,\dots,M$, $m=0,1,\dots,n$, that is $4KN_p$, where N_p are the number of points of $\mathcal{X}$ contained in p and $K=(M+1)(M+2)/2$. Thus, considering a block p at a generic level l, we have

$$c_1\le 4\frac{N}{4^{l+1}}K,$$

Let c_2 be the cost of lines $2-27$ in Algorithm 1, that is the cost of

$$\sum_{n=0}^{M}\sum_{m=0}^{n} d_{n,m}h_{n,m}(\boldsymbol{X}_i-\boldsymbol{Z})\sin(m\omega_{x_i-z})\mathbf{v}_{n,m}(\boldsymbol{Z}) \quad \text{and} \quad \sum_{n=0}^{M}\sum_{m=0}^{n} d_{n,m}h_{n,m}(\boldsymbol{X}_i-\boldsymbol{Z})\cos(m\omega_{x_i-z})\mathbf{w}_{n,m}(\boldsymbol{Z}), \tag{16}$$

for all $n=0,1,\dots,M$, $m=0,1,\dots,n$, $p\in P_l, l=1,2,\dots,L$, $\boldsymbol{z}$ the centre of p, $i\in S_p$, where $\mathbf{v}_{n,m},\mathbf{w}_{n,m}$ contain the contributions of the points $\boldsymbol{y}_j$ for the sine and cosine part, respectively, already calculated in (15); $\mathbf{v}_{n,m},\mathbf{w}_{n,m}$ correspond respectively to $\mathbf{v}_k,\mathbf{w}_k$ in Algorithm 1. In the following discussion, the cost of the multiplication by the factor $d_{n,m}$ is neglected since such factor can be included in $h_{n,m}$ or, equivalently, in $j_{n,m}$ during the construction of the data structures. We analyse the cost c_2 as function of the level l. At the first level, we have $N/16$ points of $\mathcal{X}$ in each block $p\in P_1$ and at most there are 12 well-separated blocks from p, that are blocks in S_p. Thus, the cost $c_2^{(1)}$ at the first level is

$$c_2^{(1)}\le 4K12\frac{N}{16}=3KN,$$

where K is defined above and gives the number of addenda in each sum appearing in (16). At level $l\ge 2$, each block of the level $l-1$ is divided into 4 blocks, so fixing the block p of the first level, p is divided into 4^{l-1} blocks at level l. For each small block there are at most 27 well-separated blocks. Thus, the cost $c_2^{(l)}$ at level l is

$$c_2^{(l)}\le 4K4^{l-1}27\frac{N}{4^{l+1}}=\frac{27}{4}KN.$$

Since in the costs $c_2^{(l)}, l=1,2,\dots,L$, we fixed a block of the first level and we referred to it during the calculations, by considering also that there are 16 different blocks at the first level, we have

$$c_2\le 16\sum_{l=1}^{L} c_2^{(l)}\le 108KNL.$$

Let c_3 be the cost of lines $28-35$ in Algorithm 1, that is the cost of

$$\sum_{j\in p} A_{i,j}\mathbf{u}_j,$$

for all $i\in NS_p$ and $p\in P_L$. With analogous arguments to the ones used above, i.e., the number of blocks in the first level, the number of blocks obtained at level L from the subdivision of a first-level block into smaller blocks and the number of points into a block, considering also that the maximum number of non well-separated blocks is at most 9, we obtain

$$c_3\le 16\,4^{L-1}\frac{N}{4^{L+1}}9\frac{N}{4^{L+1}}=9\frac{N^2}{4^{L+1}}.$$

So, the total cost of the procedure is

$$\begin{aligned} c =& c_1 + c_2 + c_3 \leq \\ \leq& N\left(\frac{4K}{4^{l+1}} + 108KL + \frac{9N}{4^{L+1}}\right) \leq \\ \leq& N\left(109KL + \frac{9N}{4^{L+1}}\right). \end{aligned} \tag{17}$$

In formula (17), the parameters K and N are fixed before the procedure, whereas the number of levels L should be chosen in order to obtain the minimum c. Now, if we consider the upper bound for c, we should search for the minimum of this upper bound. In more detail, let $\alpha = 109K, \beta = 9N/4, g(\lambda) = \alpha\lambda + \frac{\beta}{4^\lambda}$, where λ is the real extension of the integer variable L. Then, it can be shown that g has a minimum in $\lambda_{min} = \log_4\left(\frac{\beta \log_{10}(4)}{\alpha}\right)$, where $g(\lambda_{min}) = \frac{\alpha}{\log_{10}(4)}\left(\log_{10}\left(\frac{\beta \log_{10}(4)}{\alpha}\right) + 1\right)$. A similar result can be obtained without the extension of the integer variable L, because of the convexity of the function g. In conclusion, the upper bound of c is proportional to $N \log_{10}(N)$.

3 Numerical simulations

We present a numerical experiment performed with the technique described in the previous section, in order to show the accuracy and the efficiency of the proposed method. The experiment consists in computing the product of the matrix A and a random vector whose components range is $[-1, 1]$, by using two different techniques: the standard matrix-vector multiplication, and the translation technique described in Section 2.

We fix the shape parameter $t = 1$. We note that the value of the shape parameter does not affect the theoretical basis of the method as described in section 2. However, it can slightly influence the convergence of the method and in turn the computational cost analysed in the previous section. In particular, the parameter t appears only in the quantities $j_{n,m}$ and $h_{n,m}$, which can be calculated outside the procedure and stored in appropriate matrices. From standard arguments in RBF literature, the shape parameter can significantly affect the conditioning of the linear system. From preliminary numerical experiments, not reported in this paper, the proposed method only slightly affects the system conditioning. A deeper analysis is required to precisely state this result.

We consider the set of quasi-random N Halton points [15] in $\Omega = [0, 1]^2$, with $N = 2(4), 4(4), 6(4), 8(4), 1(5)$, where $x(y)$ denotes the real number $x \cdot 10^y$. Furthermore, in the translation technique, the truncation index in (12) is fixed to $M = 10$ and the maximum number of levels is $L = 1, 2, 3$. The results are reported in Table 1 and Figure 3. Table 1 shows the elapsed time T in the computation of the product of A and a random vector by using a standard row-column product, the elapsed time in the same computation by using the translation technique, the absolute error E in infinity-norm between the result computed by the translation technique and the one obtained by the standard row-column product, and the number of levels L that gives the best efficiency result among $L = 1, 2, 3$. In addition, Figure 3 reports a comparison of the execution times of the different methods. In the $x-$axis the number of interpolation points is reported, while the vertical axis gives the normalised execution time, that is the execution time divided by the number of interpolation points, i.e., $\frac{T}{N}$. The line with circular markers represents the usual matrix-vector multiplication, while the line with squared markers represents the proposed technique. In Figure 3, the substantial reduction of the execution time can be easily appreciated, as well as the logarithmic trend of the proposed strategy.

The results in Table 1 and Figure 3 have been obtained on a Workstation equipped with an Intel(R) Xeon(R) CPU E5-2620 v3 @2.40GHz, operative system Red Hat Enterprise Linux, release 7.5. All computations have been made in double precision and the FORTRAN codes have been compiled by the NAGWare f95 Compiler.

N	Standard	Translation		
	T [s]	T [s]	E	L
2(4)	1.62(0)	1.58(0)	2.67(−9)	1
4(4)	1.79(1)	5.23(0)	4.61(−9)	2
6(4)	3.21(1)	9.78(0)	6.62(−9)	2
8(4)	8.38(1)	1.58(1)	8.72(−9)	2
1(5)	1.28(2)	2.33(1)	1.06(−8)	2

Table 1: Execution times T (in seconds), relative error E and number of levels L by varying the number of points N, for the truncation index $M = 10$, where $x(y)$ denotes the real number $x \cdot 10^y$.

From these results, we can observe a substantially reduced computational time T of the translation technique with respect to the standard computation, especially for large value of N. Moreover, the error E shows the high accuracy provided by the proposed method despite the small value of the truncation index M.

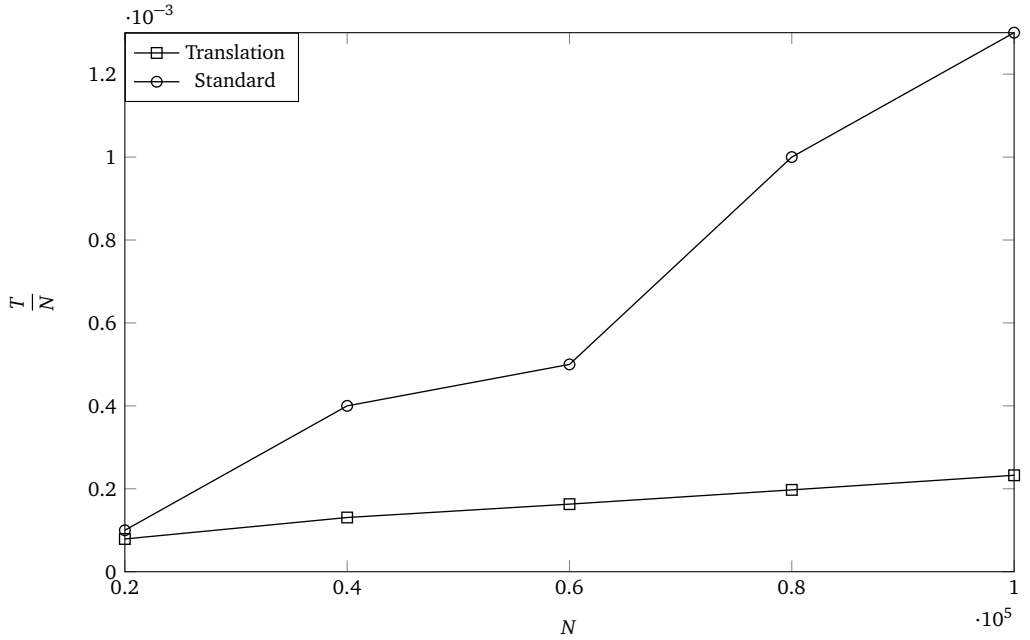


Figure 3: Trends of the normalised execution times with the two strategies, when $M = 10$

4 Conclusions

We considered a scattered interpolation problem with the IMQ-RBFs and we proposed an efficient strategy for computing the product between the coefficient matrix and a generic vector. This method is based on the well-known decomposition formula in spherical coordinates for the Laplacian operator and a simple translation technique. The presented numerical experiments show strongly promising results, both for the efficiency and the accuracy of the proposed technique. In fact, such computational strategy has a smaller computational cost than the standard matrix-vector multiplication, and the computed numerical solutions are almost the same. These results encourage further investigations about this method. In particular, such strategy should be implemented into an iterative method for the solution of the interpolation problem. An interesting future development of this work is also the generalisation of this technique to other choices of RBFs as well as to the collocation problem for the solution of differential equations. Finally, the decomposition of A could be profitably exploited for the construction of ad-hoc preconditioner for the interpolation problem.

Acknowledgments This research has been accomplished within Rete ITaliana di Approssimazione (RITA), the thematic group on "Approximation Theory and Applications" of the Italian Mathematical Union and partially funded by GNCS-INδAM.

References

[1] R. K. Beatson, J. B. Cherrie, C. T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, 11(2–3):253–270, 1999.

[2] R. K. Beatson, G. N. Newsam. Fast evaluation of radial basis functions: Moment-based methods. *SIAM Journal on Scientific Computing*, 19(5):1428–1449, 1998.

[3] R. K. Beatson, G. N. Newsam. Fast evaluation of radial basis functions: I. *Computers and Mathematics with Applications*, 24(12):7–19, 1992.

[4] R. K. Beatson, M. J. D. Powell, A. M. Tan. Fast evaluation of polyharmonic splines in three dimensions. *IMA Journal of Numerical Analysis*, 27(3):427–450, 2007.

[5] P. Benner, T. Mach. The preconditioned inverse iteration for hierarchical matrices. *Numerical Linear Algebra with Applications*, 20(1):150–166, 2013.

[6] D. Cai, E. Chow, L. Erlandson, Y. Saad, Y. Xi. SMASH: Structured matrix approximation by separation and hierarchy. *Numerical Linear Algebra with Applications*, 25(6):e2204, 2018.

[7] J. C. Carr, W. R. Fright, R. K. Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE transactions on medical imaging*, 16(1):96–107, 1997.

[8] H. S. Carslaw, J. C. Jaeger. Conduction of heat in solids. *Clarendon press*, 1959.

[9] J. B. Cherrie, R. K. Beatson, G. N. Newsam. Fast Evaluation of Radial Basis Functions: Methods for Generalized Multiquadrics in $\mathbb{R}^n$. *SIAM Journal on Scientific Computing*, 23(5):154–1571, 2002.

[10] N. Egidi, P. Maponi. The efficient solution of direct medium problems by using translation techniques. *Mathematics and Computers in Simulation*, 79(8):2361–2372, 2009.

[11] J. Flusser. An adaptive method for image registration. *Pattern Recognition*, 25(1):45–54, 1992.

[12] C. Franke, R. Schaback. Solving partial differential equations by collocation using radial basis functions. *Applied Mathematics and Computation*, 93(1):73–82, 1998.

[13] A. Gillman, P.M. Young, P.G. Martinsson. A direct solver with O (N) complexity for integral equations on one-dimensional domains. *Frontiers of Mathematics in China*, 7(2):217–247, 2012.

[14] L. Greengard, V. Rokhlin. An adaptive method for image registration. *Journal of computational physics*, 73(2):325–348, 1987.

[15] J.H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

[16] R. L. Hardy. Theory and applications of the multiquadric-biharmonic method 20 years of discovery 1968–1988. *Computers & Mathematics with Applications*, 19(8–9):163–208, 1990.

[17] D. Lazzaro, L. B. Montefusco. Radial basis functions for the multivariate interpolation of large scattered data sets. *Journal of Computational and Applied Mathematics*, 140(1–2):521–536, 2002.

[18] S. Le Borne, L. Grasedyck. H-matrix preconditioners in convection-dominated problems. *SIAM journal on matrix analysis and applications*, 27(4):1172–1183, 2006.

[19] P.G. Martinsson, V. Rokhlin. A fast direct solver for boundary integral equations in two dimensions. *Journal of Computational Physics*, 205(1):1–23, 2005.

[20] P. M. Morse, H. Feshbach. Methods of Theoretical Physics. *McGraw-Hill Book Company, inc*, 1274, 1953.

[21] T. Poggio, F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[22] M. R. Spiegel, s. Lipschutz, J. Liu. Schaum's Outlines: Mathematical Handbook of Formulas and Tables. *McGraw-Hill*, 166, 2018.

[23] Y. Xi, J. Xia, R. Chan. A fast randomized eigensolver with structured LDL factorization update. *SIAM Journal on Matrix Analysis and Applications*, 35(3):974–996, 2014.

[24] Y. Xi, J. Xia, S. Cauley, V. Balakrishnan. Superfast and stable structured solvers for Toeplitz least squares via randomized sampling. *SIAM Journal on Matrix Analysis and Applications*, 35(1):44–72, 2014.

[25] J. Xia, Y. Xi, M. Gu. A superfast structured solver for Toeplitz linear systems via randomized sampling. *SIAM Journal on Matrix Analysis and Applications*, 33(3):837–858, 2012.