© 2026 The Author(s). Published by Università degli studi di Torino. Open access under the CC BY license.

J. Approx. Softw., Vol. 3, No. 1 (2026) 3

Journal of Approximation Software

doi: 10.13135/3103-1935/12293 – ISSN: 3103–1935

A MATLAB Code for Stable Differentiation of Radial Basis Function Approximations

E. Larsson [1,⋆], A. Heryudono [2], A. Michael [1], and C. Piret [3]

[1] *Department of Information Technology, Uppsala University, Sweden*
[2] *Department of Mathematics, University of Massachusetts Dartmouth, MA, USA*
[3] *Department of Mathematical Sciences, Michigan Technological University, MI, USA*

Received: 15/07/2025 – Published: 24/02/2026
Communicated by: A. De Rossi

Abstract

Radial basis function approximation allows for fitting of scattered data and for solving partial differential equations in non-trivial geometries. This requires solving potentially ill-conditioned linear systems of equations. The RBF-QR class of methods provides a change of basis that results in well-conditioned matrices in the same approximation space. In this paper, we derive improved algorithms for differentiation in the RBF-QR basis that solves some accuracy issues of the original implementation. We provide all second derivatives in three spatial dimensions, while previously only the Laplacian was implemented. We also provide a high level interface to compute differentiation matrices using either RBF-QR or the direct evaluation method.

Keywords: radial basis function, ill-conditioning, stable method, RBF-QR (MSC2020: 65D12, 65N35)

1 Introduction

Radial basis function (RBF) approximation methods [4] are a subclass of kernel approximation methods. They are particularly well suited for the fitting of scattered data and for solving partial differential equations (PDEs) in non-trivial geometries. The simplest form of an RBF approximation can be expressed as

$$s(\underline{x}) = \sum_{j=1}^{n} \lambda_j \phi(\|\underline{x} - \underline{x}_j\|) \triangleq \sum_{j=1}^{n} \lambda_j \phi_j(\underline{x}), \tag{1}$$

where $\underline{x} \in \mathbb{R}^d$, λ_j are the expansion coefficients, $\phi(r)$ is a radial basis function, and $\phi_j(\underline{x})$ refers to a radial basis function centered at a node point $\underline{x}_j$. Requiring the RBF approximation to

⋆ Corresponding author: elisabeth.larsson@it.uu.se

interpolate given data $f_i = f(\underline{x}_i)$, $i = 1,\dots,n$ at the node points, leads to a linear system of equations

$$s(\underline{x}_i) = \sum_{j=1}^{n} \lambda_j \phi_j(\underline{x}_i) = f_i, \quad i = 1,\dots,n. \tag{2}$$

In matrix form, the problem can be written as $A\underline{\lambda} = \underline{f}$, where we call A the interpolation matrix. Applying a linear differential operator $\mathscr{L}$ to the RBF approximation (1) yields

$$\mathscr{L}s(\underline{x}) = \sum_{j=1}^{n} \lambda_j \mathscr{L}\phi_j(\underline{x}). \tag{3}$$

Evaluating (3) at a set of evaluation points $\underline{y}_i$, $i = 1,\dots,m$ leads to the matrix representation $\underline{s}_{\mathscr{L}} = B_{\mathscr{L}}\underline{\lambda}$, where $B_{\mathscr{L}} \in \mathbb{R}^{m\times n}$ is an evaluation matrix. Combining the matrix equations associated with (1) and (3) formally yields

$$\underline{s}_{\mathscr{L}} = B_{\mathscr{L}}A^{-1}f \triangleq D_{\mathscr{L}}\underline{f}. \tag{4}$$

The matrix $D_{\mathscr{L}}$ is a called a differentiation matrix and operates directly on the nodal data bypassing the coefficient vector $\underline{\lambda}$, which sometimes can attain large magnitudes.

The interpolation matrix is guaranteed to be non-singular when the node points are distinct and the RBF is positive definite [18]. In the case of conditionally positive definite RBFs, non-singularity is ensured by augmenting the approximation (1) with a low-degree polynomial basis and by imposing an additional constraint on the RBF coefficients in (2), [14]. A selection of commonly used RBFs, along with their key properties, is presented in Table 1.

Table 1: Selected RBFs with abbreviation and definition.

Positive definite RBFs		
Name	$\phi(r)$	
Gaussian (GS)	$e^{-(\varepsilon r)^2}$	
Inverse quadratic (IQ)	$\frac{1}{1+(\varepsilon r)^2}$	
Inverse multiquadric (IMQ)	$\frac{1}{\sqrt{1+(\varepsilon r)^2}}$	
Conditionally positive definite RBFs of order m		
Name	Definition	m
Multiquadric (MQ)	$\sqrt{1+(\varepsilon r)^2}$	0
Polyharmonic spline (PHS)	r^{2p+1}	$p = 0,1,\dots$
Thin plate spline (TPS)	$r^{2p}\log r$	$p = 1,2,\dots$

The most commonly used infinitely smooth RBFs contain a shape parameter ε, which by scaling the argument r, controls the shape of the function from flat to steep. The shape parameter has a significant influence on the accuracy of the RBF approximation. For smoother data, in the sense of having a rapidly decaying Fourier series, the best result is achieved for smaller values of ε, [10]. However, for small values of ε the basis functions become nearly flat, and hence nearly linearly dependent. The condition number of the interpolation matrix is proportional to ε^{-2K}, where $K \geq 1$ increases with the number of node points n, [3, 10, 17].

Because the condition number of the interpolation matrix grows exponentially as the shape parameter decreases, direct computation or inversion becomes numerically unstable for small ε. In contrast, the condition number of the corresponding differentiation matrix remains bounded. This discrepancy reflects a poor choice of basis within an otherwise well-suited approximation

space for smooth data. The basic idea of the family of RBF-QR methods is to expand the nearly flat RBFs in a better basis $\{V_j(x)\}_{j=0}^{\infty}$, such that

$$\underline{\phi}(x) = CD\underline{V}(x),$$

where $\underline{\phi}(x) = [\phi_1(x), \ldots, \phi_n(x)]^T$, C is a matrix of coefficients with magnitude of $\mathcal{O}(1)$, D is a diagonal matrix that isolates the harmful scaling due to the shape parameter, and $\underline{V}(x) = [V_1(x), \ldots]^T$. The key step in the method involves applying a QR-factorization to the coefficient matrix $C = QR$. To better explain the matrix manipulations, we rewrite $R = [R_1 \; R_2]$ and $D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$ where R_1 and D_1 both are $n \times n$ matrices. Since D is diagonal with increasing powers of ε as entries, the product $RD = [R_1 D_1 \;\; R_2 D_2]$ has columns that are scaled by increasing powers of ε. However, since R is upper triangular, so is RD, hence its rows also are scaled by increasing powers of ε. Those powers of ε can thus safely be factored out of RD to the left, which is crucial in what follows.

Rather than expressing the interpolant as $s(x) = \underline{\lambda}^T \underline{\phi}(x)$, we seek a numerically stable basis $\underline{\psi}(x)$, spanning the same space, such that $s(x) = \underline{\beta}^T \underline{\psi}(x)$, where $\underline{\psi}(x)$ resembles $\underline{V}(x)$ up to an ε-dependent perturbation. That is, we want $\underline{\psi}(x) = [I \;\; \tilde{R}]\underline{V}(x)$, where the entries of $\tilde{R}$ are scaled by increasing powers of ε:

$$\begin{aligned}
s(x) &= \underline{\lambda}^T \phi(x) \\
&= \underline{\lambda}^T Q \, R \, D \, V(x) \\
&= \underbrace{(\lambda^T Q)(R_1 D_1)}_{\underline{\beta}^T} [I \;\; D_1^{-1} R_1^{-1} R_2 D_2] V(x) \\
&= \underline{\beta}^T [I \;\; \underbrace{D_1^{-1} R_1^{-1} R_2 D_2}_{\tilde{R}}] V(x) \\
&= \underline{\beta}^T \; \underline{\psi}(x).
\end{aligned}$$

The harmful scaling effect from D_1^{-1} can now be handled by analytically computing $\tilde{D}$ with elements $\tilde{D}_{ij} = d_{jj}/d_{ii}$ and then computing $\tilde{R} = \tilde{D} \odot R_1^{-1} R_2$, where $\odot$ denotes the Hadamard (elementwise) product. The infinite expansion is truncated where the elements become small relative a tolerance (currently the machine precision). The expansion typically converges rapidly for small values of ε, but may not be practically useful or even convergent for larger values.

The first RBF-QR algorithm was developed for the surface of the sphere [7], and used expansions in spherical harmonics from [8]. The expansions are available for the MQ, IMQ, IQ, and GS infinitely smooth RBFs. The second type of RBF-QR algorithm was derived for one, two, and three space dimensions [6], but only for GS RBFs. The initial expansion step that is used is a Taylor expansion in the shape parameter that is dimension agnostic, but that results in a basis close to the monomials, which is not stable enough. Therefore, subsequent dimension dependent transformation steps are added involving Chebyshev polynomials in the radial direction and trigonometric/spherical harmonic functions in the angular directions.

A more elegant expansion for GS RBFs is achieved in [5] using the Mercer expansion in terms of orthogonal eigenfunctions. However, as closed-form expressions for these eigenfunctions are available only in one dimension, the multivariate case relies on tensor product constructions, leading to a larger number of expansion functions in the multivariate case than is strictly necessary. The method also has some additional parameters to tune.

Another approach for GS RBFs based on expansions using the generating function of the Hermite polynomials is derived in [9]. This method bears similarity to the algorithm in [5], but places particular emphasis on the treatment of anisotropic basis functions and the development of suitable truncation criteria.

The focus of this paper is the stable and accurate evaluation of derivatives of RBF approximations. For small shape parameter values we employ the RBF-QR method from [6]. An initial version of the algorithms, along with accompanying MATLAB software for computing derivatives in two and three spatial dimensions, was presented in [11]. The present work introduces new capabilities and improved algorithms, with the main updates summarized as follows:

- Expressions for and implementation of all second order derivatives in three dimensions are added (previously only the Laplacian was available).
- An algorithm for accurate evaluation for points (exponentially) close to the edge of the expansion domain is provided.
- A unified interface for computing RBF differentiation matrices with the direct method and with the RBF-QR method is introduced.

The Matlab implementation of the algorithms can be found at https://github.com/elisabethl/rbfdiff.

The structure of the paper is as follows: Section 2 presents closed-form expressions for the RBF-QR expansions and their derivatives. The subtleties arising from the use of Legendre and Chebyshev polynomials are explored in Sections 3 and 4, respectively. The Matlab software implementation is discussed in Section 5. Section 6 provides validation of the new algorithms, followed by additional illustrative examples in Section 7. Finally, Section 8 discusses the method's limitations and potential avenues for future extensions.

2 The RBF-QR expansions and their derivatives

A generic expansion function in radial, polar, or spherical coordinates can be written

$$V(\underline{x}) = f(r) \begin{cases} \operatorname{sgn}(\underline{x}) & \text{in one dimension,} \\ g(\theta) & \text{in two dimensions,} \\ Y(\theta,\varphi) & \text{in three dimensions,} \end{cases}$$

where (r,θ) are polar coordinates in two dimensions and (r,θ,φ) are spherical coordinates in three dimensions. The radial function is given by $f(r) = e^{-\varepsilon^2 r^2} r^{2m} T_{j-2m}$, where $T_j(r)$ is the jth Chebyshev polynomial of the first kind. For the one-dimensional case, $m \equiv 0$. The angular part in two dimensions is given by

$$g(\theta) = \begin{cases} \cos((2m+p)\theta) \\ \sin((2m+p)\theta), \end{cases} \quad 2m+p \neq 0,$$

and the angular functions in three dimensions are $Y(\theta,\phi) = Y_\mu^\nu(\theta,\varphi)$, $\mu = 2m+p$, $\nu = -\mu,\ldots,\mu$, where Y_μ^ν is a real spherical harmonic function, and $p = \operatorname{mod}(j,2)$.

The gradients and Laplacians in two and three dimensions were derived in [11] as well as all the second derivatives in two dimensions. However, the individual derivatives of order two in three dimensions were left out due to a more complicated interaction between singular terms.

Here, we provide also the second derivatives in three dimensions. We group the terms of the derivatives such that each radial and angular part is non-singular as a whole. The general form of the second derivatives in three dimensions becomes

$$\begin{aligned}\frac{\partial^2 V}{\partial\alpha\partial\beta} &= c_0^1 f'' Y_\mu^\nu + c_0^2 \frac{f'}{r} Y_\mu^\nu \\ &\quad + c_1^1 \frac{f'}{r}\frac{\partial Y_\mu^\nu}{\partial\varphi} + c_1^2 \frac{f'}{r}\frac{1}{\sin\theta}\frac{\partial Y_\mu^\nu}{\partial\varphi} + c_1^3 \left(\frac{f'}{r} - \frac{f}{r^2}\right)\frac{\partial Y_\mu^\nu}{\partial\theta} \\ &\quad + c_2^1 \frac{f}{r^2}\left(\frac{1}{\sin^2\theta}\frac{\partial^2 Y_\mu^\nu}{\partial\varphi^2} + \frac{\cos\theta}{\sin\theta}\frac{\partial Y_\mu^\nu}{\partial\theta}\right) + c_2^2 \frac{f}{r^2}\frac{\partial^2 Y_\mu^\nu}{\partial\theta\partial\varphi} \\ &\quad + c_2^3 \frac{f}{r^2}\left(\frac{1}{\sin^2\theta}\frac{\partial Y_\mu^\nu}{\partial\varphi} - \frac{\cos\theta}{\sin\theta}\frac{\partial^2 Y_\mu^\nu}{\partial\theta\partial\varphi}\right) + c_2^4 \frac{f}{r^2}\frac{\partial^2 Y_\mu^\nu}{\partial\theta^2},\end{aligned}$$

where $c_i^j(\alpha,\beta)$ are trigonometric functions in θ and φ. The specific coefficient functions for the individual derivatives are given in Tables 2, 3, and 4.

Table 2: Coefficient functions for the zero order terms in the second derivatives.

$\alpha\beta$	c_0^1	c_0^2
xx	$\cos^2\varphi\sin^2\theta$	$\sin^2\varphi+\cos^2\varphi\cos^2\theta$
yy	$\sin^2\varphi\sin^2\theta$	$\cos^2\varphi+\sin^2\varphi\cos^2\theta$
zz	$\cos^2\theta$	$\sin^2\theta$
xy	$\sin\varphi\cos\varphi\sin^2\theta$	$-\sin\varphi\cos\varphi\sin^2\theta$
xz	$\cos\varphi\sin\theta\cos\theta$	$-\cos\varphi\sin\theta\cos\theta$
yz	$\sin\varphi\sin\theta\cos\theta$	$-\sin\varphi\sin\theta\cos\theta$

Table 3: Coefficient functions for the first order terms in the second derivatives.

$\alpha\beta$	c_1^1	c_1^2	c_1^3
xx	$-2\sin\varphi\cos\varphi$	0	$-2\cos^2\varphi\sin\theta\cos\theta$
yy	$2\sin\varphi\cos\varphi$	0	$2\sin^2\varphi\sin\theta\cos\theta$
zz	0	0	$-2\sin\theta\cos\theta$
xy	$\cos^2\varphi-\sin^2\varphi$	0	$2\sin\varphi\cos\varphi\sin\theta\cos\theta$
xz	0	$-\sin\varphi\cos\theta$	$\cos\varphi(\cos^2\theta-\sin^2\theta)$
yz	0	$\cos\varphi\cos\theta$	$\sin\varphi(\cos^2\theta-\sin^2\theta)$

Table 4: Coefficient functions for the second order terms in the second derivatives.

$\alpha\beta$	c_2^1	c_2^2	c_2^3	c_2^4
xx	$\sin^2\varphi$	0	$2\sin\varphi\cos\varphi$	$\cos^2\varphi\cos^2\theta$
yy	$\cos^2\varphi$	0	$-2\sin\varphi\cos\varphi$	$\sin^2\varphi\cos^2\theta$
zz	0	0	0	$\sin^2\theta$
xy	$-\sin\varphi\cos\varphi$	0	$-\cos^2\varphi+\sin^2\varphi$	$\sin\varphi\cos\varphi\cos^2\theta$
xz	0	$\sin\varphi$	0	$-\cos\varphi\sin\theta\cos\theta$
yz	0	$-\cos\varphi$	0	$-\sin\varphi\sin\theta\cos\theta$

The expansion functions for $m = 0$, and hence $\mu = p$ become special cases and need to be derived separately. We have

$$\frac{\partial^2 V}{\partial\alpha\partial\beta} = c_0^1 f'' Y_0^0 + c_0^2 \frac{f'}{r} Y_0^0, \quad m = 0, \quad p = 0,$$

and

$$\frac{\partial^2 V}{\partial\alpha\partial\beta} = c_0^1 f'' Y_1^\nu + \left(\frac{f'}{r} - \frac{f}{r^2}\right)\left(c_0^2 Y_1^\nu + c_1^1 \frac{\partial Y_1^\nu}{\partial\varphi} + c_1^2 \frac{1}{\sin\theta}\frac{\partial Y_1^\nu}{\partial\varphi} + c_1^3 \frac{\partial Y_1^\nu}{\partial\theta}\right),$$

for $m = 0$, $p = 1$, and $\nu = -1, 0, 1$.

3 Evaluating expressions involving Legendre polynomials

When computing the derivatives of the spherical harmonic part of the expansion functions as described in the previous subsection, we need to operate on the normalized associated Legendre polynomials and evaluate expressions that in a naive implementation would result in division by zero. Here, we go through the different recursion formulas that are used in the numerical implementation.

The real spherical harmonics that we use are given by

$$\begin{aligned} Y_\mu^\nu(\theta,\varphi) &= \frac{1}{\sqrt{2\pi}} P_\mu^{|\nu|}(\cos\theta) g_\nu(\varphi), \qquad \mu > 0, \\ Y_0^0 &= \frac{1}{\sqrt{2\pi}}, \end{aligned}$$

where $g_v(\varphi)$ and its companion function $h_\nu(\varphi)$ are given by

$$g_\nu(\varphi) = \begin{cases} \cos(\nu\varphi), & \nu = 0, \cdots, \mu, \\ \sin(|\nu|\varphi), & \nu = -\mu, \cdots, -1, \end{cases}$$

$$h_\nu(\varphi) = \begin{cases} \sin(\nu\varphi), & \nu = 0, \cdots, \mu, \\ \cos(|\nu|\varphi), & \nu = -\mu, \cdots, -1. \end{cases}$$

We define the following functions derived from the Legendre polynomials

$$\begin{aligned} Q_\mu^\nu &= \frac{P_\mu^\nu}{\sin\theta}, & S_\mu^\nu &= \frac{\partial P_\mu^\nu}{\partial\theta}, \\ R_\mu^\nu &= \frac{P_\mu^\nu}{\sin^2\theta}, & T_\mu^\nu &= \frac{\partial^2 P_\mu^\nu}{\partial\theta^2}. \end{aligned}$$

We also introduce two special combinations

$$\begin{aligned} A_\mu^\nu &= -\frac{\nu^2}{\sin^2\theta} P_\mu^\nu + \frac{\cos\theta}{\sin\theta} S_\mu^\nu, \\ B_\mu^\nu &= -\nu\left(\frac{1}{\sin^2\theta} P_\mu^\nu - \frac{\cos\theta}{\sin\theta} S_\mu^\nu\right). \end{aligned}$$

We can express the relevant derivatives of the spherical harmonics in terms of these functions

$$\begin{aligned}
\frac{\partial Y_\mu^\nu}{\partial\varphi} &= -\frac{\nu}{\sqrt{2\pi}}P_\mu^{|\nu|}(\cos\theta)h_\nu(\varphi), & \frac{1}{\sin\theta}\frac{\partial Y_\mu^\nu}{\partial\varphi} &= -\frac{\nu}{\sqrt{2\pi}}Q_\mu^{|\nu|}(\cos\theta)h_\nu(\varphi),\\
\frac{\partial^2 Y_\mu^\nu}{\partial\varphi^2} &= -\frac{\nu^2}{\sqrt{2\pi}}P_\mu^{|\nu|}(\cos\theta)g_\nu(\varphi),
\end{aligned}$$

$$\begin{aligned}
\frac{\partial Y_\mu^\nu}{\partial\theta} &= \frac{1}{\sqrt{2\pi}}S_\mu^{|\nu|}(\cos\theta)g_\nu(\varphi), & \frac{\partial^2 Y_\mu^\nu}{\partial\theta\partial\varphi} &= \frac{\nu}{\sqrt{2\pi}}S_\mu^{|\nu|}(\cos\theta)h_\nu(\varphi),\\
\frac{\partial^2 Y_\mu^\nu}{\partial\theta^2} &= \frac{1}{\sqrt{2\pi}}T_\mu^{|\nu|}(\cos\theta)g_\nu(\varphi),
\end{aligned}$$

$$\begin{aligned}
\frac{1}{\sin^2\theta}\frac{\partial^2 Y_\mu^\nu}{\partial\varphi^2}+\frac{\cos\theta}{\sin\theta}\frac{\partial Y_\mu^\nu}{\partial\theta} &= \frac{1}{\sqrt{2\pi}}A_\mu^{|\nu|}(\cos\theta)g_\nu(\varphi),\\
\frac{1}{\sin^2\theta}\frac{\partial Y_\mu^\nu}{\partial\varphi}-\frac{\cos\theta}{\sin\theta}\frac{\partial^2 Y_\mu^\nu}{\partial\theta\partial\varphi} &= \frac{1}{\sqrt{2\pi}}B_\mu^{|\nu|}(\cos\theta)h_\nu(\varphi).
\end{aligned}$$

For the Legendre polynomials themselves and the Legendre polynomials divided by powers of $\sin\theta$, we use the following recursion:

$$\begin{aligned}
F_{\mu,j}^{\mu} &= \tfrac{(2\mu)!}{2^\mu\mu!}(-\sin\theta)^{\mu-j},\\
F_{\mu,j}^{\mu-1} &= -\tfrac{\cos\theta}{\sin\theta}F_{\mu,j}^{\mu},\\
F_{\mu,j}^{\nu} &= -\tfrac{1}{(\mu+\nu+1)(\mu-\nu)}\left(2(\nu+1)\tfrac{\cos\theta}{\sin\theta}F_{\mu,j}^{\nu+1}+F_{\mu,j}^{\nu+2}\right).
\end{aligned}$$

The boundary values $\cos\theta=\pm1$ are handled separately and are given by $F_{\mu,0}^0(\pm1)=(\pm1)^\mu$, $F_{\mu,1}^1(\pm1)=-1/2(\pm1)^{\mu-1}\mu(\mu+1)$, and $F_{\mu,2}^2(\pm1)=1/8(\pm1)^{\mu-2}(\mu-1)\mu(\mu+1)(\mu+2)$. Using the normalization factor $C=\frac{(-1)^\nu}{(\mu-\nu)!(\mu+\nu)!}\sqrt{\mu+\frac{1}{2}}$ we have $P_\mu^\nu=CF_{\mu,0}^\nu$, $Q_\mu^\nu=CF_{\mu,1}^\nu$, and $R_\mu^\nu=CF_{\mu,2}^\nu$.

The derivatives with respect to θ follow a different recursion formula, and we have

$$\begin{aligned}
S_\mu^\mu &= -\sqrt{\tfrac{\mu}{2}}P_\mu^{\mu-1}, & T_\mu^\mu &= -\sqrt{\tfrac{\mu}{2}}S_\mu^{\mu-1},\\
S_\mu^0 &= \sqrt{\mu(\mu+1)}P_\mu^1, & T_\mu^0 &= \sqrt{\mu(\mu+1)}S_\mu^1,\\
S_\mu^\nu &= -\tfrac{1}{2}\left(a_\mu^\nu P_\mu^{\nu-1}-b_\mu^\nu P_\mu^{\nu+1}\right), & T_\mu^\nu &= -\tfrac{1}{2}\left(a_\mu^\nu S_\mu^{\nu-1}-b_\mu^\nu S_\mu^{\nu+1}\right), \quad |\nu|=1,\ldots,\mu-1,
\end{aligned}$$

where $a_\mu^\nu=\sqrt{(\mu+\nu)(\mu-\nu+1)}$ and $b_\mu^\nu=\sqrt{(\mu+\nu+1)(\mu-\nu)}$.

The two combination terms need to be rewritten such that the singular terms are removed analytically. The recursion formulas then become

$$\begin{aligned}
A_\mu^\mu &= -\mu^2R_\mu^\mu+\cos\theta\sqrt{\frac{\mu}{2}}Q_\mu^{\mu-1},\\
A_\mu^0 &= -\cos\theta\sqrt{\mu(\mu+1)}Q_\mu^1,\\
A_\mu^\nu &= -\nu(\nu-1)R_\mu^\nu-\nu P_\mu^\nu-b_\mu^\nu\cos\theta Q_\mu^{\nu+1},
\end{aligned}$$

with the special case $A_1^1=-P_1^1$, and

$$\begin{aligned}
B_\mu^\mu &= -\mu\left(\frac{P_\mu^\mu}{\sin^2\theta}-\frac{\cos\theta}{\sin\theta}\sqrt{\frac{\mu}{2}}P_\mu^{\mu-1}\right),\\
B_\mu^0 &= 0,\\
B_\mu^\nu &= \nu(\nu-1)R_\mu^\nu-\nu^2P_\mu^\nu+-\nu b_\mu^\nu\cos\theta Q_\mu^{\nu+1}.
\end{aligned}$$

4 Accurate evaluation of the radial part involving Chebyshev polynomials

For a sum of Chebyshev polynomials

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{n-1} c_k T_k(x)$$

the regular Clenshaw recurrence is

$$\begin{aligned} u_k(x) &= 2xu_{k+1}(x) - u_{k+2}(x) + a_k, \\ f(x) &= xu_1(x) - u_2(x). \end{aligned}$$

We want to use this for evaluation of the basis functions, so we consider the case $a_j = 1$, $a_k = 0$, $k \neq j$, and we use the initial condition $u_{j+2} = u_{j+1} = 0$. To take a derivative, we differentiate the equations above to get

$$\begin{aligned} u_k'(x) &= 2xu_{k+1}'(x) - u_{k+2}'(x) + 2u_{k+1}(x), \\ f'(x) &= xu_1'(x) - u_2'(x) + u_1(x). \end{aligned}$$

Note that u_k takes the role of the coefficients here. We can take a second derivative to get

$$\begin{aligned} u_k''(x) &= 2xu_{k+1}''(x) - u_{k+2}''(x) + 4u_{k+1}'(x), \\ f''(x) &= xu_1''(x) - u_2''(x) + 2u_1'(x). \end{aligned}$$

This method of evaluating the derivatives of the Chebyshev polynomials is accurate for all values of x, and especially for values near the end points. A different and less stable formulation was used in the previous implementation of the RBF-QR code [11].

Even with accurately computed Chebyshev polynomials and derivatives, direct evaluation of the function

$$\tilde{R}_j(x) = \frac{T_j - xT_j'}{x^2}, \qquad j = 3,5,\ldots,\infty, \tag{5}$$

leads to very large cancellation errors. As $x \to 0$, the two values that are subtracted approach j/x, leading to a worst case rounding error of $2\varepsilon_M j/x$, where ε_M is the machine precision.

The function can be written down explicitly as

$$\tilde{R}_j = (-1)^{(j-1)/2} \sum_{k=1}^{\frac{j-1}{2}} x^{2k-1} \frac{(-1)^{k+1} 2k}{(2k+1)!} j \prod_{i=1}^{k} (j^2 - (2i-1)^2). \tag{6}$$

This expression allows for accurate evaluation for small values of x, but not for all x. There is however no cancellation in (5) for large values of x. Hence we can switch expressions at a suitable breakpoint. The rounding error is of the order $5e-13$ at $x = 0.04$, and from numerical experiments, this is the optimal breakpoint for j up to at least $j_{\max} = 50$.

The expression (6) can be stably evaluated through a Horner like scheme. The core algorithm is shown in Listing 1. At most 10 terms are used, corresponding to `nmax=19` as further terms are small enough that they do not influence the numerical result.

Code 1: The scheme for computing $\tilde{R}_j$

```
nterm = min(nmax,jend); % Must be an odd number
%
Rs = (nterm-1)/factorial(nterm);
for k=nterm-2:-2:1
    Rs = -Rs.*(x.^2*(jvec.^2-k^2)) + (k-1)/factorial(k);
end
Rs = Rs.*(1./x*(jvec.*(-1).^jhalf)); % Alternating signs
```

5 The Matlab software implementation

The latest version of the Matlab implementation of the methods described in this paper can be found at https://github.com/elisabethl/rbfdiff. The `tests` folder also includes the previous version of the two- and three-dimensional RBF-QR implementations for comparison purposes. The previous implementation of the RBF-QR code in two dimensions was translated into Julia by Fredrik Fryklund, KTH Royal Institute of Technology and can be found at https://github.com/fryklund/rbfqr. An overview of the current code structure is shown in Figure 1.

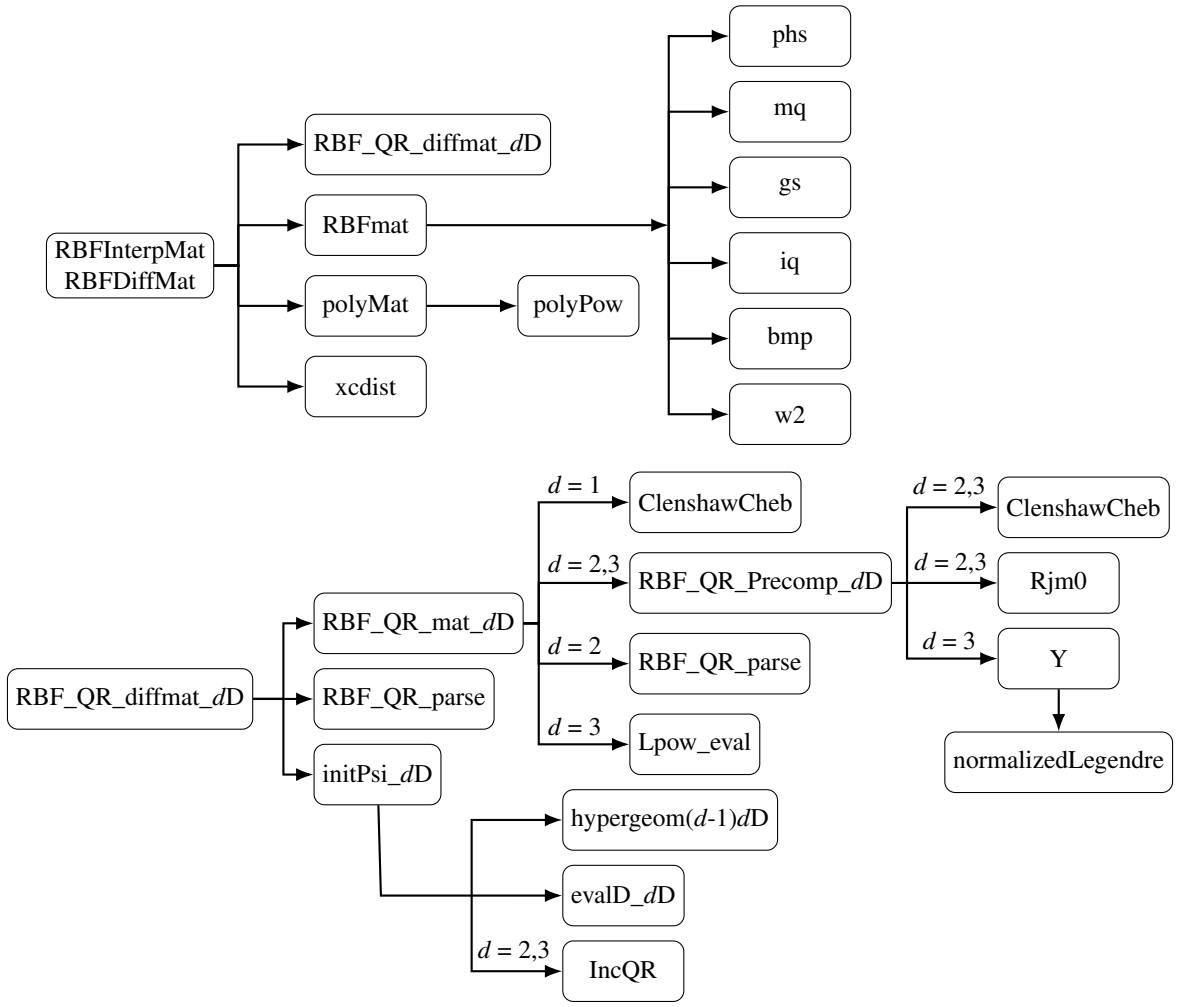

Figure 1: The dependency structure of the different parts of the software. Dependencies for `RBFInterpMat` and `RBFDiffMat` are equivalent. The dimension is given by d and unless otherwise indicated, the dependency holds for all dimensions.

At the top level, we only expose the two subroutines `RBFInterpMat` that sets up an RBF interpolation matrix, and `RBFDiffMat` that computes the desired differentiation matrices as in (4) using direct evaluation (RBF-direct) or RBF-QR. These routines allow for adding a polynomial basis to the RBF basis for RBF-direct, but not for RBF-QR, as in the small shape parameter limit the RBF approximation is close to polynomial in itself [3, 10, 17].

If we want to use, e.g., Hermite collocation, we need to build the matrices in a different way. Therefore, we also provide interfaces for directly computing differentiation matrices or interpolation and evaluation matrices. The available calls can be found in Listing 2.

Code 2: The available calls. For calls that contain "dD", there are three different versions: 1D, 2D, and 3D.

```
%
% Calls to compute and factorize the RBF interpolation matrix A.
%
Psi = RBFInterpMat(phi,pdeg,ep,xc,xe);
Psi = RBFInterpMat(phi,pdeg,ep,C,R);

Psi = RBF_QR_diffmat_dD(ndiff,xc,ep,C,R);
[A,Psi,Te] = RBF_QR_diffmat_dD(ndiff,xc,ep,xe);

[Psi]=InitPsi_dD(ep,xc);
%
% Call to compute interpolation or evaluation matrices using RBF-direct
%
A=RBFmat(phi,ep,r,op,opdim);
%
% Calls to compute interpolation or evaluation matrices using RBF-QR
%
[A,Te]=RBF_QR_mat_dD(Psi,op,xe);
[A,Te]=RBF_QR_mat_dD(Psi,op,Te);
%
% Calls to compute differentiation matrices given an interpolation structure Psi
%
[D,Te] = RBFDiffMat(ndiff,Psi,xe);
[D,Te] = RBFDiffMat(ndiff,Psi,Te);

[D,Te] = RBF_QR_diffmat_dD(ndiff,Psi,xe);
[D,Te] = RBF_QR_diffmat_dD(ndiff,Psi,Te);
```

The input and output parameters can in some cases have different meanings for different types of call, and here we summarize what they are in each case.

`phi, char(:)` is an abbreviation of the desired type of RBF, such as `'gs','mq','phs'` for RBF-direct and `'rbfqr'` for RBF-QR.

`pdeg, int` is the degree of the added monomial basis for RBF-direct, where `pdeg=-1` is used for not having any polynomial terms, but is unused for RBF-QR.

`ep, real(:)` is the real shape parameter. For RBF-direct it is possible to give a vector of shape parameter values for the different basis functions, while for RBF-QR, the shape parameter should be one value for all basis functions.

`xc, real(:,dim)` is an array of cordinates for the node points. If `Init_Psi_dD` is called directly, then `xc` needs to be given in polar or spherical coordinates within the expansion domain for 2D and 3D, respectively.

`xe, real(:,dim)` is an array of cordinates for the evaluation points. If `RBF_QR_mat_dD` is called directly, then `xe` needs to be given in polar or spherical coordinates within the expansion domain for 2D and 3D, respectively.

`C, real(1,dim), R, real` are the desired center and radius of the expansion domain in the coordinate system where `xc` and `xe` are defined.

`ndiff, int` defines which derivatives to compute, with `0` for evaluation, `1` for the gradient, `1.5` for the Laplacian, `2` for the Hessian, and `ndiff`$= 2s-0.5$ for Δ^s.

`op, char(:)` describes a scalar differential operator and the different routines currently use different names with some examples being `'0','1','2','x','xy','m2','L','L4'`.

`opdim, int(:)` contains information about which dimension, e.g., a first derivative should be applied to.

`Te, struct` is a structure that is used for storing precomputed information from previous calls if several differentiation matrices are computed for the same `xe`.

`Psi, struct` is a structure that mainly contains the factorized interpolation matrix, but it also holds information about what transformations to apply to the evaluation points, and which polynomial degree and shape parameter to use.

`A, (cell array of) real(:,:)` is either a single interpolation, evaluation or differentiation matrix or, in the case of the gradient or the Hessian, a cell array of matrices.

`D, (cell array of) real(:,:)` is either a single differentiation matrix or, in the case of the gradient or the Hessian, a cell array of differentiation matrices.

Remark 5.1. Note that either `R` and `C`, or `xe`, have to be provided when constructing `Psi` in the RBF-QR case. The reason is that evaluation at points outside the expansion domain is not well-defined. Hence, the approximation domain needs to be defined explicitly through `R` and `C` or implicitly through `xc` and `xe`, such that all points in the approximation domain fall into the expansion domain.

5.1 User extensions

Users can customize and contribute to the software in, at least, the following ways:

- The code can easily be extended to handle more types of RBFs by implementing a new function `abbrv.m`, where `abbrv` is a short name for the new type, containing the function and derivative expressions.
- New operators can also be introduced. For RBF-direct, they should be added to all types of RBFs. For RBF-QR, they require separate (and potentially complicated) derivations and implementations for 1D, 2D, and 3D. The operators should then be added to `RBF_QR_parse` that keeps track of operator-related information.

6 Validation of the code

The implementation comes with a number of test scripts that aim to validate the correctness of the algorithms from the mathematical, numerical, and implementation perspectives. Here, we reference these scripts and provide a selection of plots with comments.

6.1 The Legendre polynomials and the spherical harmonics

For the operations on modified associated Legendre polynomials described in Section 3, the function `testLegendre(m,N)` is used for validating the expressions for the derivatives and composite expressions against their finite difference approximations, where `m`=μ is the order of the polynomial and N is the number of evaluation points. The finite differences are computed using these evaluation points. Figure 2 shows two examples of the comparison. The analytical and finite differences derivatives agree well, and the agreement improves as the number of evaluation points increases.

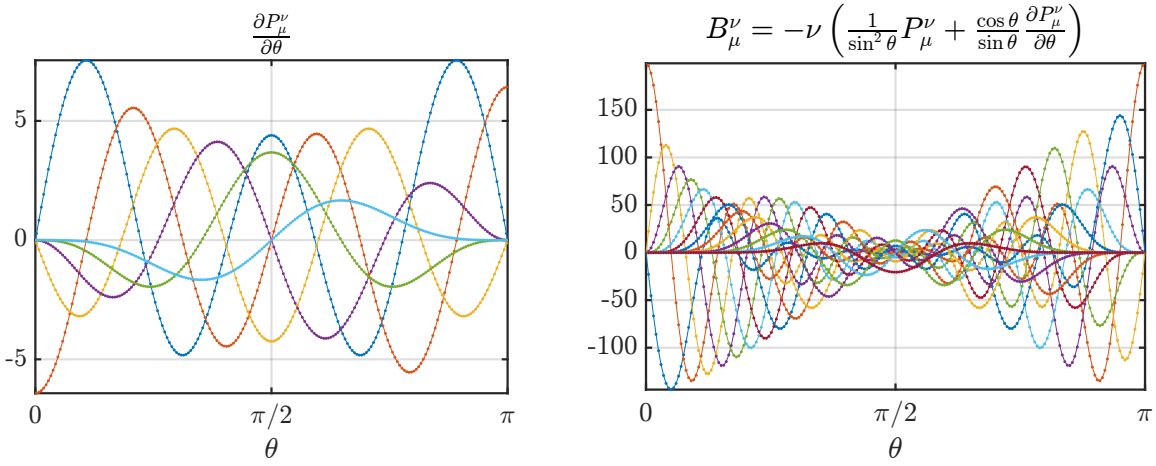

Figure 2: Comparison between evaluation of expressions involving Legendre polynomials using the recursive algorithms from Section 3 (solid lines) and using finite differences (dots). Left: The first derivative with respect to θ of P_5^ν for $\nu = 0,\ldots,5$. Right: The expression B_{14}^ν for $\nu = 0,\ldots,14$.

To ensure that the evaluation is stable near the end points of the interval $\cos\theta = \pm 1$ as well as for $\cos\theta = 0$, the function `testLegendreExtremePts(m)` plots the computed functions in loglog scale near these three points. When the order of the polynomial increases, some of the derivatives and expressions become very small, and after $\mu = 41$, the expression B reaches underflow. However, the algorithm can continue to deliver numerical results up to $\mu = 85$. This is a very high order that covers reasonable use cases. Results for these two orders are illustrated in Figure 3.

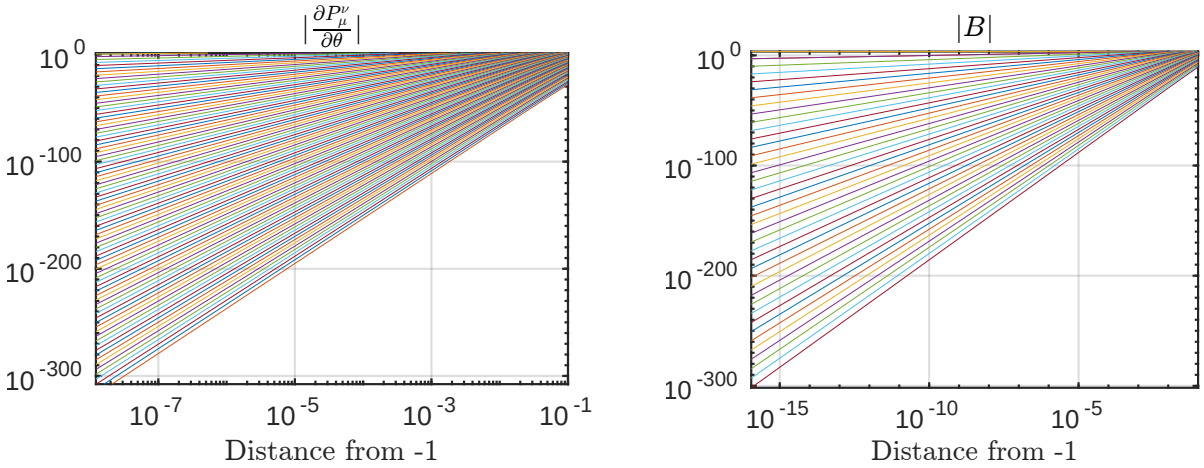

Figure 3: Left: The computed values of the first derivative in θ close to $\cos\theta = -1$ for $\mu = 85$. Below around $\cos\theta = -1 + 10^{-8}$, the algorithm returns zero. Right: The expression B is the first to reach the underflow level. At $\mu = 41$, the lowest values are right above the underflow level.

To furthermore evaluate the algorithm for computing the derivatives of the spherical harmonics, the function `testY(mu,N)` makes a comparison of the full derivatives along a line, against finite differences. The norm differences are also computed and displayed. As N increases, the difference decreases, as the finite difference approximation improves. Figure 4 shows two examples of derivatives.

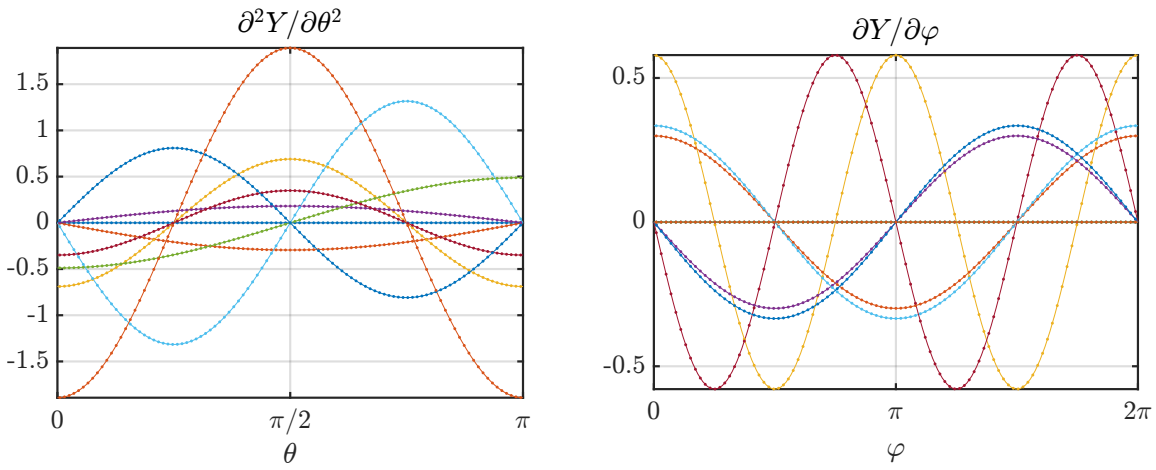

Figure 4: Comparison between the evaluated spherical harmonics derivatives (solid lines) and the finite difference approximations (dots) for $\mu = 0, 1, 2$ and $\nu = -\mu, \ldots, \mu$.

6.2 The radial functions involving Chebyshev polynomials

We first compare the previous and current implementation of the first and second derivatives of the Chebyshev polynomials near $r = 1$ using the script `testRadialFun2D`. The difference between the new (correct) value and the result from the previous algorithm is shown in Figure 5. The previous algorithm yields the same value for all $r \lesssim 10^{-10}$, which is why the difference changes character there. For the second derivative, there is a jump at $10\varepsilon_M$ due to a tolerance and the use of the limit below the tolerance. In these experiments, we can see that the errors in the previous algorithm are large, but they would in most cases go unnoticed because the probability of points falling precisely in the sensitive region is small.

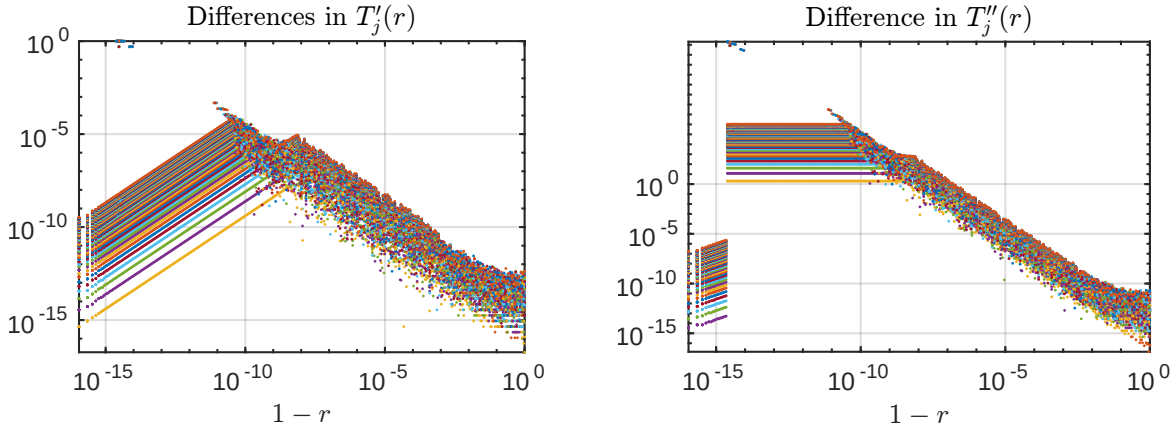

Figure 5: The magnitude of the difference between the previous implementation of the derivatives of the Chebyshev polynomials and the result from the Clenshaw algorithm for $j_{\max} = 50$.

In the second experiment, we consider the evaluation of the radial function (5). In the left part of Figure 6, the worst case cancellation error follows the theoretical prediction closely, and in the right part of the figure, we see the difference between the two evaluation methods. Note that the difference has a local minimum. Below that the cancellation error grows, but above it, the recursive evaluation (6) becomes inaccurate.

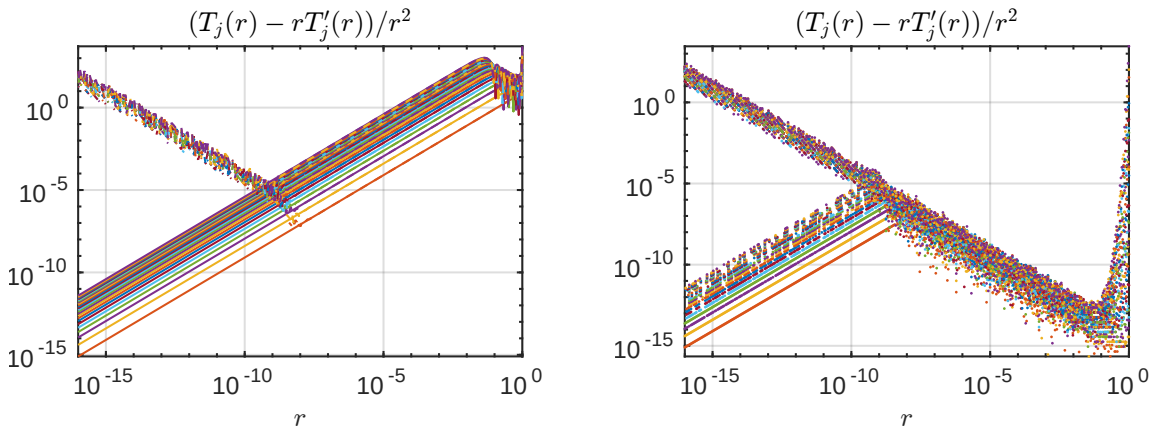

Figure 6: Comparison between direct evaluation of $R(r)$ using (5) (dashed lines) and recursive evaluation using (6) (solid lines). The left subfigure shows the values, and the right subfigure shows the absolute values of the differences for $j_{\max} = 50$.

6.3 Validation of the improvement in accuracy

A relevant question is how much the effects of the improved numerical accuracy of the underlying computations affect the errors in the differentiation of an RBF approximation. The scripts `testAccuracy2D` and `testAccuracy3D` compute the errors in the RBF interpolant and in the Laplacian of the interpolant using the original RBF-QR implementation and the new RBF-QR implementation. The errors in the interpolant are very similar for the two implementations, whereas the errors in the Laplacian have significant differences. The errors are computed along r for fixed angular coordinates, and are shown near $r = 0$ and near $r = 1$ in Figures 7 and 8.

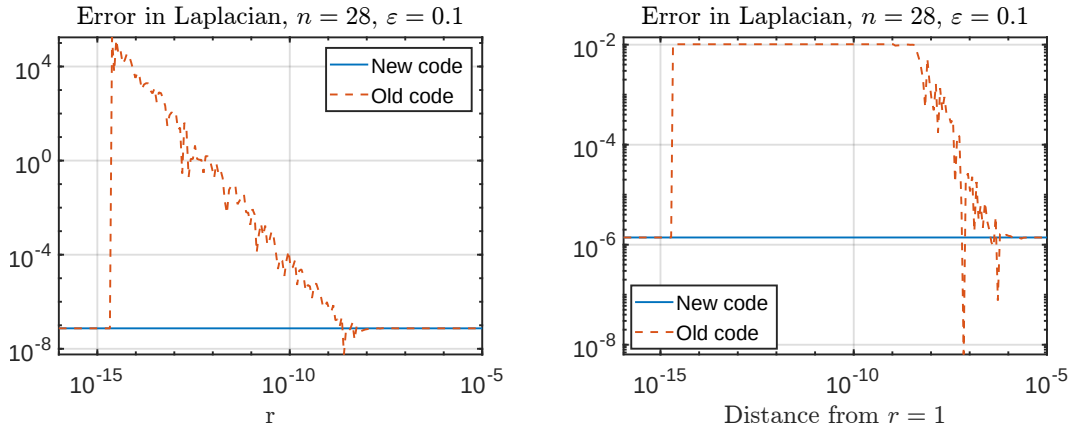

Figure 7: The magnitude of the errors in the Laplacian approximation for the new and old implementations of RBF-QR in two spatial dimensions near the extreme points in r.

In two spatial dimensions, we see significant differences near both points, while only $r = 1$ displays issues for the three dimensional case. Since quasi random or even fully random evaluation points may be used by a meshfree method, the method needs to be accurate for all cases, and the new implementation ensures this.

We also investigate whether the range of problem sizes that can be addressed has increased. In Figure 9, we plot the relative error in the function value, the first derivatives, and the second derivatives of an interpolant of a smooth function in two space dimensions as the number of node points n increases. We see that the error decreases with increasing n until around $n = 400$, where we reach the rounding error limit. That is, the RBF-QR algorithm is stable for

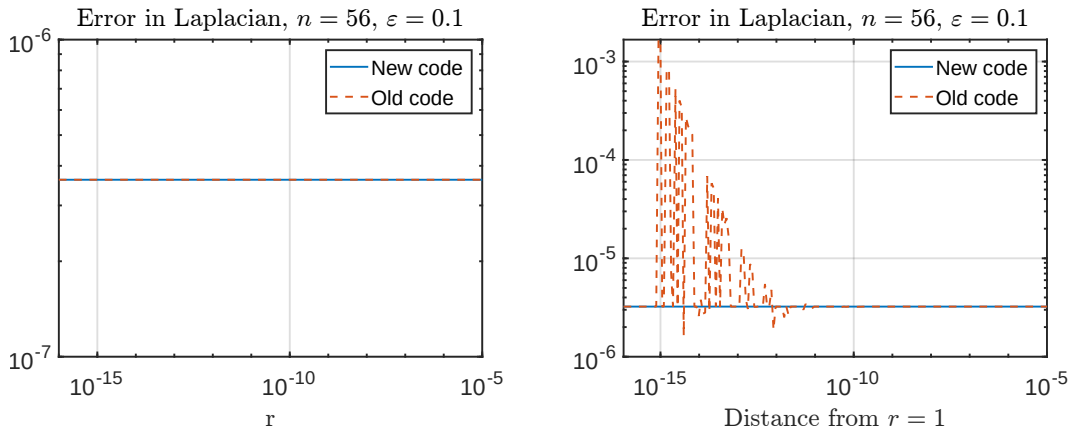

Figure 8: The magnitude of the errors in the Laplacian approximation for the new and old implementations of RBF-QR in three spatial dimensions near the extreme points in r.

large numbers of points, but the lowest error level that can be reached numerically increases with large n. It is however clear from the figure that the new implementation is significantly more accurate than the old when we go to extreme problem sizes ($n = 4000$ corresponds to a polynomial degree 88 in the flat limit). Note that in order to have good results when n increases, the node points must be clustered towards the boundary [6, 15], which is done in this experiment, see the test script `testOldNewN.m`.

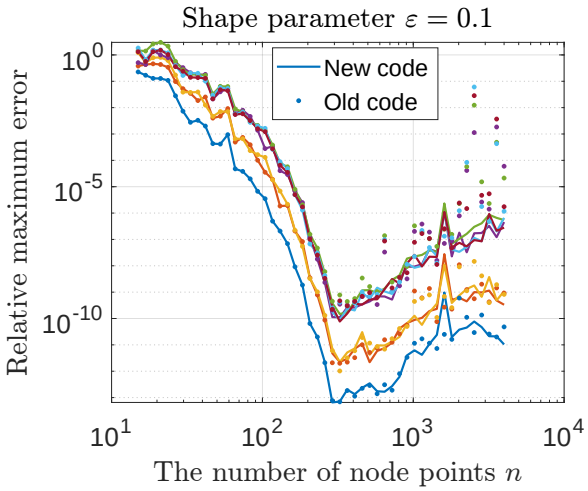

Figure 9: The interpolation and differentiation error for $f(x) = \sin(xy + y^2) - \cos(x^2)$, with shape parameter $\varepsilon = 0.1$ and with different numbers of Halton node points in the unit disc. The three groups of curves are function errors (smallest), errors in the first derivatives, and errors in the second derivatives (largest).

7 Examples of use cases

We provide examples of how the routines can be used to solve the Poisson problem with Dirichlet boundary conditions

$$\begin{aligned} \Delta u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega, \end{aligned} \tag{7}$$

where $\Omega \subset \mathbb{R}^2$ is a circle with radius $R = 1$ centred at the origin. The subroutines `RBFInterpMat` and `RBFDiffMat` are used to construct differentiation matrices for function evaluation and for the Laplacian operator over scattered nodes. We show examples of using both the RBF-FD and RBF partition of unity (RBF-PU) methods in the collocation setting [1, 2, 16, 19, 20] and in the unfitted least squares settings [12, 21]. For the RBF-FD methods we use the PHS RBF, $\phi(r) = r^3$, augmented by polynomials of degree p and for the RBF-PU methods we use the Gaussian RBF, $\phi = e^{-(\varepsilon r)^2}$ evaluated using the RBF-QR algorithm. The smooth solution function used for computing the forcing term and boundary conditions of (7) is shown in Figure 11 and is defined as

$$u_E(x,y) = \sin(2\pi xy). \tag{8}$$

We generate node points and evaluation points using the space filling Halton sequence. The resulting points for the different methods are shown in Figure 10.

For the collocation RBF-FD method (C-RBF-FD) node points $\underline{x}_j,\ j = 1,\dots,n$ are placed in the domain Ω and on the boundary $\partial\Omega$. For the unfitted least-squares RBF-FD (LS-RBF-FD) method we place the evaluation points $\underline{y}_i,\ i = 1,\dots,m$ in the domain and the node points $\underline{x}_j,\ j = 1,\dots,n$ extend outside the domain but with at least half the points in each stencil being inside the domain Ω. In both RBF-FD methods, stencils with $n_s = 2\binom{p+d}{d}$ points are used.

For the RBF-PU methods we first generate P overlapping circular patches $\{\Omega_i\}_{i=1}^P$ with centres on a Cartesian grid and radius R. In our examples we use relative overlap $\delta = 0.4$, meaning that all patches are overlaped by at least δR. We initially keep the patches with centres inside the domain and subsequently extend the patches close to the boundary such that the domain and boundary are fully covered $\bar{\Omega} \subset \cup_{i=1}^P \bar{\Omega}_j$ and the boundary of the extension is at least a distance of δR from $\partial\Omega$. Additionally, in the unfitted least squares RBF-PU (LS-RBF-PU) method evaluation points are placed in Ω but node points are generated separately inside each patch and are local to that patch. All patches include n_{loc} local points resulting in $n = n_{loc}P$ node points in total. For the collocation RBF-PU (C-RBF-PU) method different patches can, however, include a different number of local points n_{loc}.

To discuss the error from the approximation we introduce a fill distance measure for the quasi uniform node point set

$$h = \max_i \min_{i \neq j} \left\| \underline{x}_i - \underline{x}_j \right\|_2,$$

where $\|\cdot\|_2$ is the Euclidean distance. Additionally, a relative approximation error is computed for all our examples as

$$e_{\text{rel}} = \left(\frac{\sum_{i=1}^m (u(\underline{y}_i) - u_E(\underline{y}_i))^2}{\sum_{i=1}^m u(\underline{y}_i)^2} \right)^{1/2},$$

where u is the numerical solution to (7) and u_E the exact solution from (8). Note that for the collocation methods the error is evaluated at the node points, which is expected to be favourable.

A convergence study for the four methods is shown in Figure 11. For the RBF-FD methods we use a PHS basis with polynomials of degree $p = 4$ and a stencil size of $n_s = 30$, while for the unfitted LS-RBF-PU method we generate $n_{loc} = 15$ points in each patch which support a polynomial of degree 4 in 2D. Alternatively, we generate a total of $n_{loc}P$ points for the C-RBF-PU method. In all cases a least squares fit provides a slope to approximate the convergence order, which we, based on theoretical results, expect to be at least $p-1$ for all methods. Additionally, the errors when using the collocation methods are larger than when using the unfitted methods which is expected (see [12], [21]).

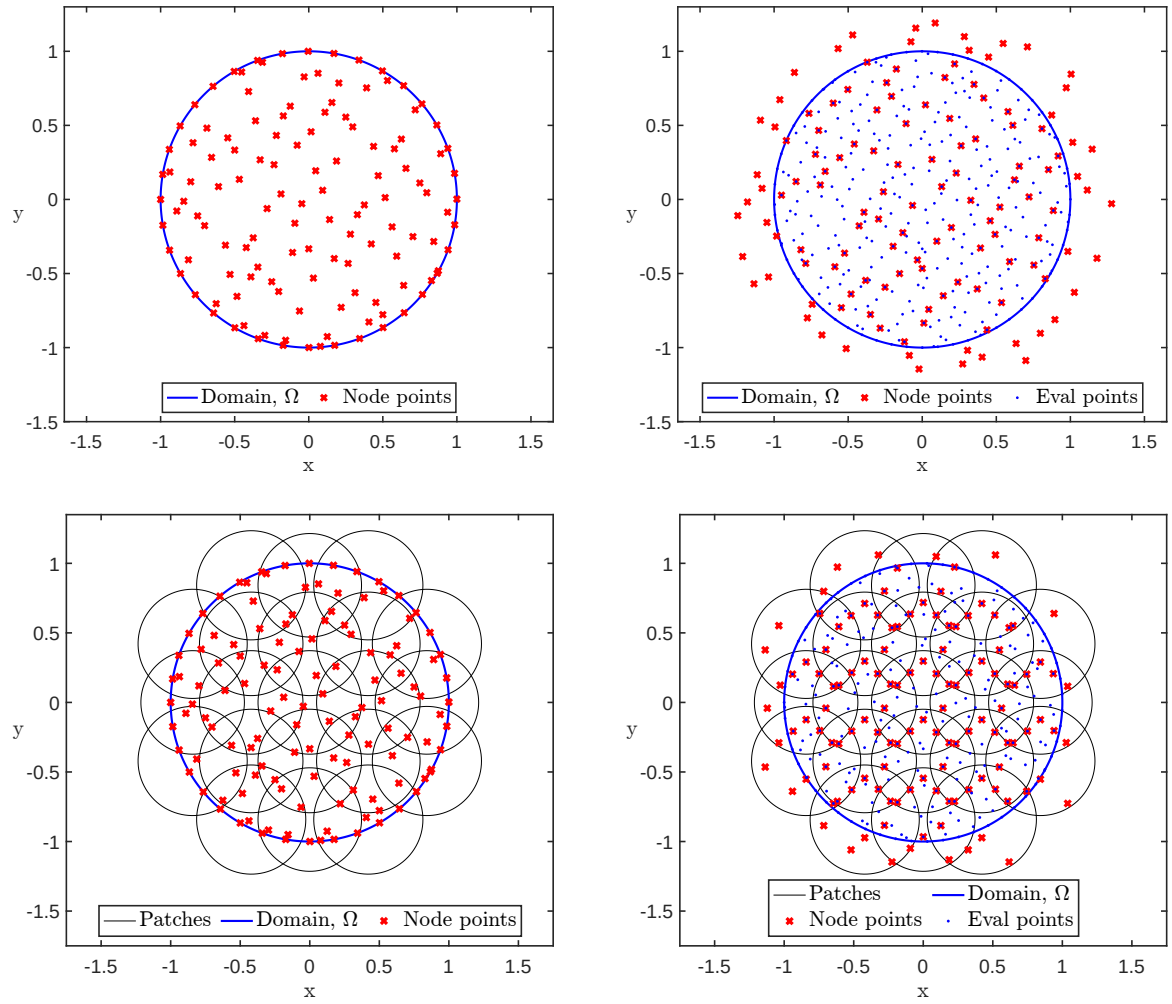

Figure 10: The point sets used to discretize and numerically solve (7) using the collocation RBF-FD method (top left), the unfitted least squares RBF-FD method (top right), the collocation RBF-PU method (bottom left) and the unfitted least squares RBF-PU method (bottom right). In all cases the number of node points is $n = 126$ and for the unfitted methods there are $m = 300$ evaluation points.

In Figure 12 we provide the error $u - u_E$ computed over the whole domain with the same fill distance for all methods.

For a more extensive example, where the second order RBF-QR derivatives in three dimensions are fully exploited, see the preprint [13].

8 Conclusions

The validation experiments performed for the new evaluation algorithms shows that the new version of the code is stable for all evaluation points. It is more stable and more accurate for large numbers of node points than the previous implementation. It also provides new functionality in terms of the second derivatives for RBF-QR in three dimensions. The unified interface makes it easier to compare RBF-direct and RBF-QR as well as approximations with or without added polynomial basis functions. There are still improvements to make regarding structure and efficiency. It is possible for users to add more functionality in terms of new basis functions

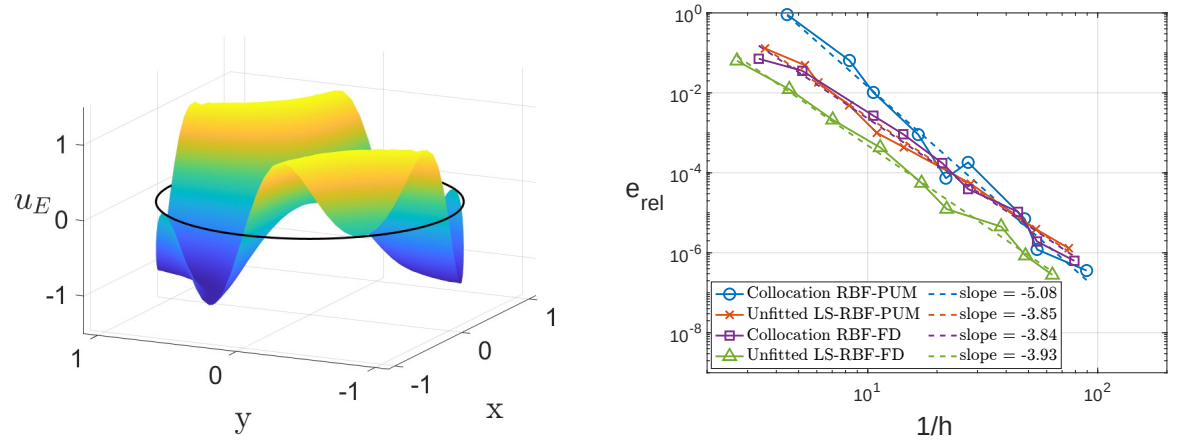

Figure 11: The smooth manufactured solution defined in (8) and used to validate our numerical results (left). The numerical error against the inverse fill distance for the C-RBF-PU, unfitted LS-RBF-PU, C-RBF-FD and unfitted LS-RBF-FD methods (right). The PHS basis was augmented with polynomials of degree $p = 4$ for the RBF-FD methods and we used $n_{loc} = 15$ for the RBF-PU methods.

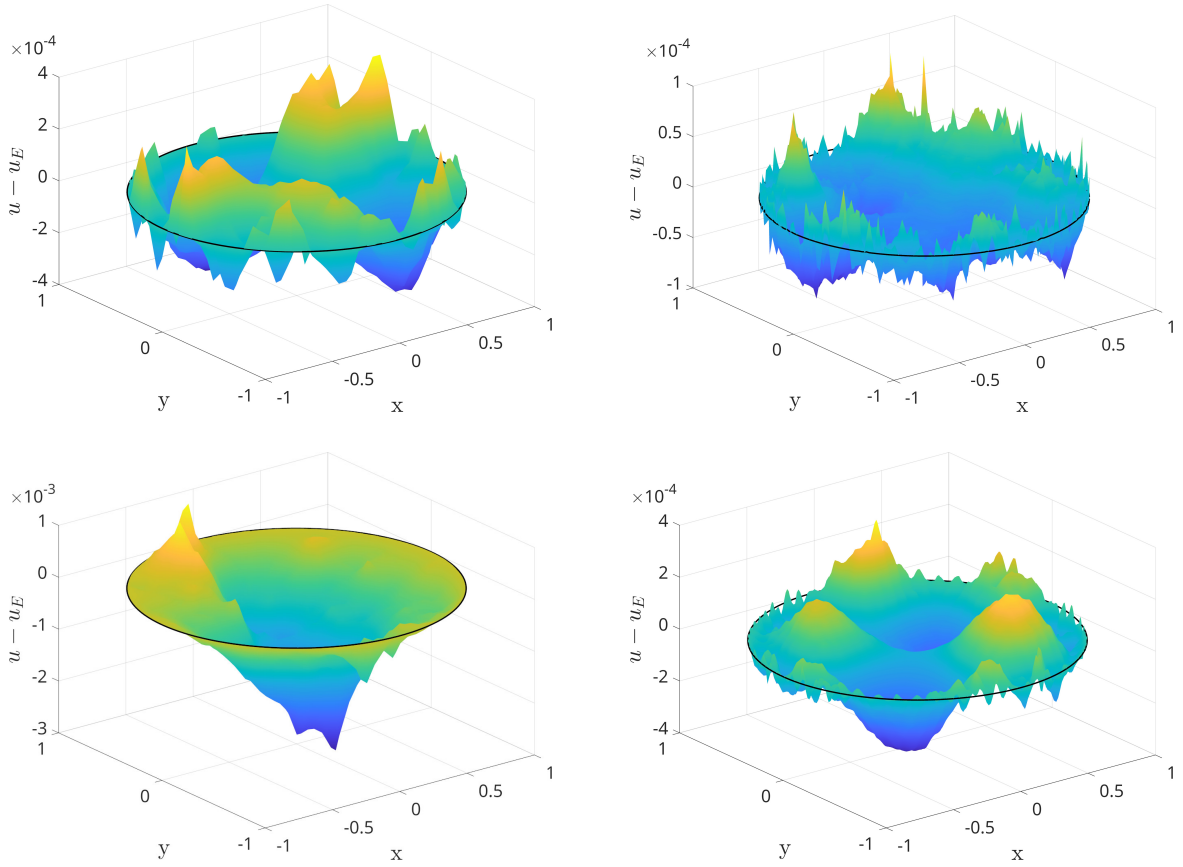

Figure 12: The error $u - u_E$ computed over the whole domain for the C-RBF-FD method (top left), the unfitted LS-RBF-FD method (top right), the C-RBF-PU method (bottom left) and the unfitted RBF-PU method (bottom right). In all cases the fill distance between centre points is $h = 0.05$.

and new operators. The implementation is already useful for us in our research projects, and we hope that it will also be useful for others in research or practical applications.

Acknowledgments

Larsson and Michael were supported by the Swedish Research Council under grant number 2020-03488. Heryudono was supported by the National Science Foundation DMS-2012011.

The RBF-QR methods [7] and [6] were conceived within Piret's and Larsson's collaborations with Bengt Fornberg, University of Colorado, Boulder. Each of the authors of this paper have contributed to different parts of the software and the paper, but there is also a fifth software contributor Erik Lehto, formerly at Uppsala University, currently not in academia, whose contributions we want to acknowledge here.

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.

References

[1] V. Bayona, N. Flyer, B. Fornberg, and G. A. Barnett, *On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs*, J. Comput. Phys., vol. 332 (2017) 257–273, URL http://dx.doi.org/10.1016/j.jcp.2016.12.008

[2] R. Cavoretto and A. De Rossi, *Error indicators and refinement strategies for solving Poisson problems through a RBF partition of unity collocation scheme*, Appl. Math. Comput., vol. 369 (2020) 124824, 18, URL http://dx.doi.org/10.1016/j.amc.2019.124824

[3] T. Driscoll and B. Fornberg, *Interpolation in the limit of increasingly flat radial basis functions*, Computers & Mathematics with Applications, vol. 43, no. 3 (2002) 413–422, URL http://dx.doi.org/10.1016/S0898-1221(01)00295-4

[4] G. E. Fasshauer, *Meshfree approximation methods with MATLAB*, vol. 6 of *Interdisciplinary Mathematical Sciences*, World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ (2007), URL http://dx.doi.org/10.1142/6437

[5] G. E. Fasshauer and M. J. McCourt, *Stable evaluation of Gaussian radial basis function interpolants*, SIAM J. Sci. Comput., vol. 34, no. 2 (2012) A737–A762, URL http://dx.doi.org/10.1137/110824784

[6] B. Fornberg, E. Larsson, and N. Flyer, *Stable computations with Gaussian radial basis functions*, SIAM J. Sci. Comput., vol. 33, no. 2 (2011) 869–892, URL http://dx.doi.org/10.1137/09076756X

[7] B. Fornberg and C. Piret, *A stable algorithm for flat radial basis functions on a sphere*, SIAM J. Sci. Comput., vol. 30, no. 1 (2007) 60–80, URL http://dx.doi.org/10.1137/060671991

[8] S. Hubbert and B. Baxter, *Radial Basis Functions for the Sphere*, in *Progress in Multivariate Approximation*, edited by J. Levesley, W. A. Light, and J. C. Mason, Birkhäuser, vol. 137 of *International Series of Numerical Mathematics*, 33–47 (2001), URL http://dx.doi.org/10.1007/978-3-0348-8272-9_4

[9] K. Kormann, C. Lasser, and A. Yurova, *Stable interpolation with isotropic and anisotropic Gaussians using Hermite generating function*, SIAM J. Sci. Comput., vol. 41, no. 6 (2019) A3839–A3859, URL http://dx.doi.org/10.1137/19M1262449

[10] E. Larsson and B. Fornberg, *Theoretical and computational aspects of multivariate interpolation with increasingly flat radial basis functions*, Computers & Mathematics with Applications, vol. 49, no. 1 (2005) 103–130, URL http://dx.doi.org/10.1016/j.camwa.2005.01.010

[11] E. Larsson, E. Lehto, A. Heryudono, and B. Fornberg, *Stable computation of differentiation matrices and scattered node stencils based on Gaussian radial basis functions*, SIAM J. Sci. Comput., vol. 35, no. 4 (2013) A2096–A2119, URL http://dx.doi.org/10.1137/120899108

[12] E. Larsson, V. Shcherbakov, and A. Heryudono, *A least squares radial basis function partition of unity method for solving PDEs*, SIAM Journal on Scientific Computing, vol. 39, no. 6 (2017) A2538–A2563, URL http://dx.doi.org/10.1137/17M1118087

[13] E. Larsson, P.-F. Villard, I. Tominec, U. Sundin, A. Michael, and N. Cacciani, *An RBF partition of unity method for geometry reconstruction and PDE solution in thin structures*, arXiv (2024), URL https://arxiv.org/abs/2403.01486

[14] C. A. Micchelli, *Interpolation of scattered data: Distance matrices and conditionally positive definite functions*, Constructive Approximation, vol. 2, no. 1 (1986) 11–22, URL http://dx.doi.org/10.1007/BF01893414

[15] R. B. Platte, L. N. Trefethen, and A. B. J. Kuijlaars, *Impossibility of fast stable approximation of analytic functions from equispaced samples*, SIAM Rev., vol. 53, no. 2 (2011) 308–318, URL http://dx.doi.org/10.1137/090774707

[16] A. Safdari-Vaighani, A. Heryudono, and E. Larsson, *A Radial Basis Function Partition of Unity Collocation Method for Convection—Diffusion Equations Arising in Financial Applications*, J. Sci. Comput., vol. 64, no. 2 (2015) 341–367, URL http://dx.doi.org/10.1007/s10915-014-9935-9

[17] R. Schaback, *Multivariate Interpolation by Polynomials and Radial Basis Functions*, Constructive Approximation, vol. 21, no. 3 (2005) 293–317, URL http://dx.doi.org/10.1007/s00365-004-0585-2

[18] I. J. Schoenberg, *Metric Spaces and Positive Definite Functions*, Transactions of the American Mathematical Society, vol. 44, no. 3 (1938) 522–536, URL http://www.jstor.org/stable/1989894

[19] C. Shu, H. Ding, and K. Yeo, *Local radial basis function-based differential quadrature method and its application to solve two-dimensional incompressible Navier–Stokes equations*, Computer Methods in Applied Mechanics and Engineering, vol. 192, no. 7 (2003) 941–954, URL http://dx.doi.org/10.1016/S0045-7825(02)00618-7

[20] A. I. Tolstykh, *On using RBF-based differencing formulas for unstructured and mixed structured-unstructured grid calculations*, in *Proceedings of the 16th IMACS World Congress 2000*, École polytechnique fédérale de Lausanne (2000), CD, ISBN 3-9522075-1-9

[21] I. Tominec and E. Breznik, *An unfitted RBF-FD method in a least-squares setting for elliptic PDEs on complex geometries*, Journal of Computational Physics, vol. 436 (2021) 110283, URL http://dx.doi.org/https://doi.org/10.1016/j.jcp.2021.110283