Journal of Scientific Computing manuscript No.
(will be inserted by the editor)

Monomial augmentation guidelines for RBF-FD from accuracy vs. computational time perspective

Mitja Jančič · Jure Slak · Gregor Kosec

Received: date / Accepted: date

Abstract Local meshless methods using RBFs augmented with monomials have become increasingly popular, due to the fact that they can be used to solve PDEs on scattered node sets in a dimension-independent way, with the ability to easily control the order of the method, but at a greater cost to execution time. We analyze this ability on a Poisson problem with mixed boundary conditions in 1D, 2D and 3D, and reproduce theoretical convergence orders practically, also in a dimension-independent manner, as demonstrated with a solution of Poisson's equation in an irregular 4D domain. The results are further combined with theoretical complexity analyses and with conforming execution time measurements, into a study of accuracy vs. execution time trade-off for each dimension. Optimal regimes of order for given target accuracy ranges are extracted and presented, along with guidelines for generalization.

Keywords meshless methods · RBF-FD · Poisson's equation · n-dimensional · convergence rates · optimal order

1 Introduction

The Radial Basis Function-generated Finite Differences (RBF-FD), a local strong form mesh-free method for solving partial differential equations (PDEs) that generalizes the traditional Finite Difference Method (FDM), was first mentioned by Tolstykh [32]. Since then, the method has become increasingly popular [9], with recent uses in linear elasticity [31], contact problems [28], geosciences [8], fluid mechanics [14], dynamic thermal rating of power lines [20], advection-dominated problems [21,25], financial sector [22], etc.

RBF-FD, similarly to other mesh-free methods, relies on approximation of differential operators on scattered nodes, which is an important advantage over mesh-based methods, as node generation is considered much easier than the mesh generation. In fact, mesh generation is often the most cumbersome part of the solution procedure in traditional methods, which, especially in 3D geometries, often requires significant assistance from the user. When meshless methods were first developed, many solutions used available mesh generators for generating discretization nodes and discarding the connectivity information after the mesh had been generated [16]. Such approach is computationally wasteful, does not generalize to higher dimensions, and some authors even reported that it failed to generate distributions of sufficient

M. Jančič
"Jožef Stefan" International Postgraduate School, Jamova cesta 39, 1000 Ljubljana, Slovenia
"Jožef Stefan" Institute, E6, Parallel and Distributed Systems Laboratory, Jamova cesta 39, 1000 Ljubljana, Slovenia
E-mail: Mitja.Jancic@ijs.si

J. Slak
"Jožef Stefan" Institute, E6, Parallel and Distributed Systems Laboratory, Jamova cesta 39, 1000 Ljubljana, Slovenia
Faculty of Mathematics and Physics, University of Ljubljana, Jadranska 19, 1000 Ljubljana, Slovenia
E-mail: Jure.Slak@ijs.si

G. Kosec
"Jožef Stefan" Institute, E6, Parallel and Distributed Systems Laboratory, Jamova cesta 39, 1000 Ljubljana, Slovenia
E-mail: Gregor.Kosec@ijs.si

quality [26]. Since then, various node positioning algorithms have been proposed. Popular algorithms use iterative approaches [12,17], advancing front methods [7,18] or sphere packing methods [5]. In 2018, a pure meshless algorithm based on Poisson disk sampling [4] was introduced. Later that year, the first dimension-independent node generation algorithm that supported distributions with spatially variable density appeared [30], where the authors also demonstrated the stability of RBF-FD on scattered nodes, even for complex non-linear problems in 3D without any special treatment of stencil selection as proposed in [23]. Instead, a cluster of nearest neighboring nodes proved to be a satisfactory stencil that can also be efficiently implemented in dimension-independent code, using specialized data structures, such as k-d tree [35].

A common drawback of often used RBFs, such as Gaussians or Hardy's multiquadrics, is that they include a shape parameter that crucially affects accuracy and stability of the approximation [33]. If the shape parameter is kept constant, the method converges, but stability issues arise when computing in the standard basis, due to high condition numbers of the collocation matrices. To fix the stability issue, more sophisticated algorithms can be used, such as RBF-CP, RBF-QA, RBF-GA and others [34], but such methods sometimes add significant additional costs. A simpler solution for the stability issue is to scale the shape parameter so that the product of the shape parameter and the nodal spacing is constant. However, this can lead to local approximations that are not convergent - this phenomenon has been called lack of convergence due to stagnation errors [6]. Stagnation can be fixed by adding monomial terms that ensure consistency up to a certain order. This technique has been used together with Polyharmonic splines (PHS) as RBFs, which have an additional advantage of not having a shape parameter [3]. In addition, the order of added monomials directly effects the order of the RBF-FD approximation, effectively enabling control over the convergence rate of the RBF-FD [2]. Various successful applications of RBF-FD with PHS have since been demonstrated, both in 2D and in 3D [26,30,3]. The dimensional independence has already been noted by, e.g., Ahmad et al. [1], but the high order RBF-FD has not yet been thoroughly analyzed with computational efficiency in mind, as the authors were more focused on solving the time-dependent part of the PDE of interest.

Although the RBF-FD formulation is dimension-independent, in the sense that the same formulation can be used in 1D, 2D, 3D and higher, translating this elegant mathematical formulation and algorithms into actual efficient computer code is far from trivial. In this paper, we present a dimension-independent PDE solution procedure based on our in-house dimension-agnostic implementation [29] of RBF-FD. By dimension-agnostic implementation we refer to the fact that exactly the same code can be used to solve problems in one, two, three or more dimensional spaces, while values of parameters are optimised for each dimension separately. The paper describes all solution procedure elements in detail and presents a thorough analysis of accuracy and execution time in one, two and three dimensions, on a Poisson problem on scattered nodes with mixed boundary conditions. To fully illustrate the dimension independence, a solution of a 4-dimensional problem on an irregular domain is presented. A C++ implementation of all discussed solution elements is freely available for download [13].

The rest of the paper is organized as follows: In section 2, the RBF-FD solution procedure is presented, in section 3, the model problem is investigated, in section 4, an additional example is shown, and in section 5, the conclusions are presented.

2 RBF-FD solution procedure

In this section, the main steps of the RBF-FD solution procedure are described. First, the domain is populated with scattered nodes. Once the nodes are positioned, in each discretization node the approximation of the partial differential operator is performed, resulting in stencil weights. Finally, in the PDE discretization phase, the PDE is transformed into a system of linear equations, whose solution stands for a numerical solution of the considered PDE.

2.1 Positioning of nodes

In the node generation algorithm, candidate nodes are generated on a d-sphere in a d-dimensional space. This effectively means that the node positioning algorithm remains the same for every number of dimensions. However, some parameters, e.g. the number of candidates, can be optimized for various numbers of dimensions.

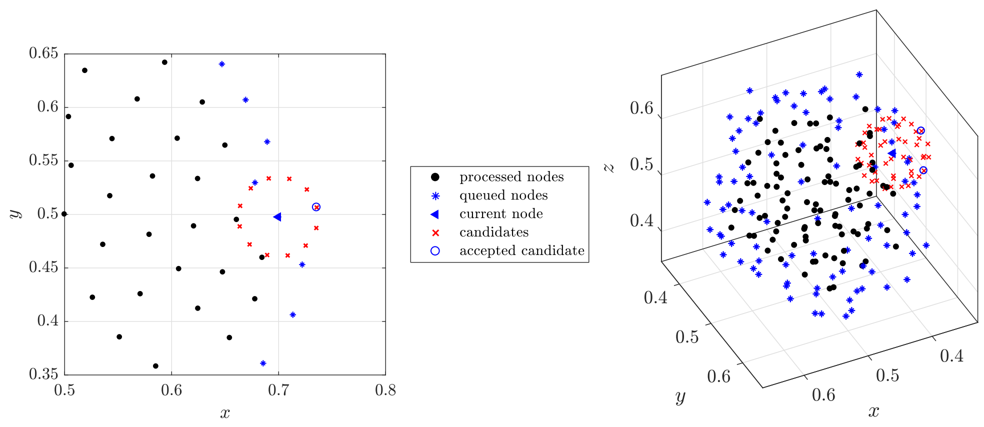

Fig. 1 Node positioning algorithm during candidate generation phase.

The node positioning algorithm takes as an input a domain $\Omega \subset \mathbb{R}^d$ with a spacing function $h\colon \Omega \to (0,\infty)$ and optionally a list of arbitrary starting "seed nodes" $X \subset \Omega$, often distributed along the boundary. It returns a set of nodes that are suitable for strong-form discretizations and distributed over Ω with mutual spacing around a point p approximately $h(p)$.

The algorithm used in this paper processes nodes in the input list in order. For each node p, a number of expansion candidates distributed uniformly on a sphere centered at p, of radius $h(p)$, are examined. If a candidate is inside the domain and sufficiently away from the already processed nodes, it is accepted and added to the list X. During the course of the algorithm, the list X is implicitly partitioned into already processed nodes, the current node, and future queued nodes. Figure 1 shows this partition at a selected iteration in 2D and 3D, along with the generated candidates from the current node, and flags the accepted ones.

Once all the elements of the list X have been processed, X is returned as the resulting set of discretization nodes. Further details and analyses of the algorithm are available in [30]. The stand-alone implementation of the algorithm is available online [27] and also included as a part of our in-house implementation of RBF-FD, the *Medusa* library [29].

2.2 Approximation of partial differential operators

Consider a partial differential operator $\mathcal{L}$ at a point $\boldsymbol{x}_c$. Approximation of $\mathcal{L}$ at a point $\boldsymbol{x}_c$ is sought using an ansatz

$$(\mathcal{L}u)(\boldsymbol{x}_c) \approx \sum_{i=1}^{n} w_i u(\boldsymbol{x}_i), \tag{1}$$

where $\boldsymbol{x}_i$ are the neighboring nodes of $\boldsymbol{x}_c$ which constitute its *stencil*, w_i are called *stencil weights*, n is the *stencil size* and u is an arbitrary function.

This form of approximation is desirable, since operator $\mathcal{L}$ at point $\boldsymbol{x}_c$ is approximated by a linear functional $\boldsymbol{w}_{\mathcal{L}}(\boldsymbol{x}_c)^\mathsf{T}$, assembled of weights w_i,

$$\mathcal{L}|_{\boldsymbol{x}_c} \approx \boldsymbol{w}_{\mathcal{L}}(\boldsymbol{x}_c)^\mathsf{T} \tag{2}$$

and the approximation is obtained using just a dot product with the function values in neighboring nodes. The dependence of $\boldsymbol{w}_{\mathcal{L}}(\boldsymbol{x}_c)^\mathsf{T}$ on $\mathcal{L}$ and $\boldsymbol{x}_c$ is often omitted, with $\boldsymbol{w}_{\mathcal{L}}(\boldsymbol{x}_c)^\mathsf{T}$ written simply as $\boldsymbol{w}$.

To determine the unknown weights $\boldsymbol{w}$, equality of (1) is enforced for a given set of basis functions. A natural choice are monomials, which are also used in FDM, resulting in the Finite Point Method [24]. However, using monomial basis suffers from potential ill conditioning [19]. An alternative approach is using an RBF basis.

In the RBF-FD discretization, the equality is satisfied for radial basis functions ϕ_j. These are RBFs, generated by a function $\phi\colon [0,\infty) \to \mathbb{R}$, centered at neighboring nodes of $\boldsymbol{x}_c$, given by

$$\phi_j(\boldsymbol{x}) = \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|). \tag{3}$$

Each ϕ_j, for $j = 1, \dots, n$, corresponds to one linear equation

$$\sum_{i=1}^{n} w_i \phi_j(\boldsymbol{x}_i) = (\mathcal{L}\phi_j)(\boldsymbol{x}_c) \tag{4}$$

for unknowns w_i. Assembling these n equations into matrix form, we obtain the following linear system:

$$\begin{bmatrix} \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_1\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_n\|) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_n\|) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} (\mathcal{L}\phi(\|\boldsymbol{x} - \boldsymbol{x}_1\|))|_{\boldsymbol{x}=\boldsymbol{x}_c} \\ \vdots \\ (\mathcal{L}\phi(\|\boldsymbol{x} - \boldsymbol{x}_n\|))|_{\boldsymbol{x}=\boldsymbol{x}_c} \end{bmatrix}, \tag{5}$$

where ϕ_j have been expanded for clarity.

The above system can be written more compactly as

$$\mathbf{A}\boldsymbol{w} = \boldsymbol{\ell}_\phi. \tag{6}$$

The matrix $\mathbf{A}$ is symmetric, and for some basis functions ϕ even positive definite [33].

Many commonly used RBFs, such as Hardy's multiquadrics or Gaussians, depend on a shape parameter, which governs their shape and consequently affects the accuracy and stability of the approximation. In this work, we use polyharmonic splines (PHS), defined as

$$\phi(r) = \begin{cases} r^k, & k \text{ odd} \\ r^k \log r, & k \text{ even} \end{cases}, \tag{7}$$

to eliminate the need for a shape parameter tuning where r denotes the Euclidean distance between two nodes. Without monomial augmentation, local approximations using only PHS are not convergent, nor do we have any guarantees of solvability. However, if the approximation given by (5) is augmented with polynomials, we obtain convergence and conditional positive definiteness, provided that the stencil nodes form a polynomially unisolvent set [33]. Augmentation is performed as follows: Let $p_1, \dots, p_s$ be polynomials forming the basis of the space of d-dimensional multivariate polynomials up to and including total degree m, with $s = \binom{m+d}{d}$. In addition to the RBF part of the approximation, an exactness constraint

$$\sum_{i=1}^{s} w_i p_j(\boldsymbol{x}_i) = (\mathcal{L}p_j)(\boldsymbol{x}_c) \tag{8}$$

for monomials, is enforced. These additional constraints make the approximation overdetermined, which is treated as a constrained optimization problem [6]:

$$\min_{\boldsymbol{w}} \left(\frac{1}{2} \boldsymbol{w}^\mathsf{T} \mathbf{A} \boldsymbol{w} - \boldsymbol{w}^\mathsf{T} \boldsymbol{\ell}_\phi \right), \text{ subject to } \mathbf{P}^\mathsf{T} \boldsymbol{w} = \ell_p. \tag{9}$$

For practical computation, the optimal solution can be expressed as a solution of a linear system

$$\begin{bmatrix} \mathbf{A} & \mathbf{P} \\ \mathbf{P}^\mathsf{T} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\ell}_\phi \\ \boldsymbol{\ell}_p \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} p_1(\boldsymbol{x}_1) & \cdots & p_s(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ p_1(\boldsymbol{x}_n) & \cdots & p_s(\boldsymbol{x}_n) \end{bmatrix}, \quad \boldsymbol{\ell}_p = \begin{bmatrix} (\mathcal{L}p_1)|_{\boldsymbol{x}=\boldsymbol{x}_c} \\ \vdots \\ (\mathcal{L}p_s)|_{\boldsymbol{x}=\boldsymbol{x}_c} \end{bmatrix}, \tag{10}$$

where $\mathbf{P}$ is a $n \times s$ matrix of polynomials evaluated at stencil nodes, $\boldsymbol{\ell}_p$ is the vector of values assembled by applying the considered operator $\mathcal{L}$ to the polynomials at $\boldsymbol{x}_c$, and $\boldsymbol{\lambda}$ are Lagrange multipliers. Weights obtained by solving (10) are taken as approximations of $\mathcal{L}$ at $\boldsymbol{x}_c$, while values $\boldsymbol{\lambda}$ are discarded. The system (10) is solvable if the stencil nodes form a polynomially unisolvent set. This could potentially be problematic near the boundary, where it might happen that all stencil nodes would be e.g. colinear or coplanar, but experience shows that this happens only with stencil sizes which are too small to be a feasible approximation. With large enough stencil sizes, stencils near the boundary always include at least some internal nodes. We did not use any special techniques to ensure unisolvency, and did not run into any unisolvency-related issues.

The exactness of (8) ensures convergence behavior and control over the convergence rate, since the local approximation has the same order as the polynomial basis used [3], while the RBF part of the approximation (5) takes care of potential ill-conditioning in purely polynomial approximation [6].

2.3 PDE discretization

Consider the boundary value problem

$$\mathcal{L}u = f \text{ in } \Omega, \tag{11}$$

$$u = g_d \text{ on } \Gamma_d, \tag{12}$$

$$\boldsymbol{n} \cdot \nabla u = g_n \text{ on } \Gamma_n, \tag{13}$$

with $\partial\Omega = \Gamma_d \cup \Gamma_n$, where the union is disjoint. The domain Ω is discretized by placing N scattered nodes $\boldsymbol{x}_i$ with quasi-uniform internodal spacing h, of which N_i are in the interior, N_d on the Dirichlet and N_n on the Neumann boundary. Additionally, N_g *ghost* or *fictitious* nodes are added outside the domain on both Neumann and Dirichlet boundary, by translating the N_d and the N_n nodes on $\partial\Omega$ for distance h in the normal direction.

In the next step, stencils $\mathcal{N}(\boldsymbol{x}_i)$ consisting of neighboring nodes are selected for each node $\boldsymbol{x}_i$. The most common approach is to compute stencils automatically, by taking n closest nodes for each node (including the node itself) as its stencil.

Next, partial differential operators appearing in the problem, such as $\mathcal{L}$ and ∂_i, are approximated at nodes $\boldsymbol{x}_i$, using the procedure described in section 2.2. The computed stencils $\boldsymbol{w}_{\mathcal{L}}$ and $\boldsymbol{w}_{\partial_i}$ are stored for later use.

For each interior node $\boldsymbol{x}_i$, the equation $(\mathcal{L}u)(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)$ is approximated by a linear equation

$$\boldsymbol{w}_{\mathcal{L}}(\boldsymbol{x}_i)^\mathsf{T}\boldsymbol{u} = \boldsymbol{f}, \tag{14}$$

where vectors $\boldsymbol{f}$ and $\boldsymbol{u}$ represent values of function f and unknowns u in stencil nodes of $\boldsymbol{x}_i$. For each Dirichlet boundary node $\boldsymbol{x}_i$, we have the equation

$$u_i = g_d(\boldsymbol{x}_i). \tag{15}$$

For Neumann boundary nodes $\boldsymbol{x}_i$, the linear equation

$$\sum_{j=1}^{d} n_j \boldsymbol{w}_{\partial_j}(\boldsymbol{x}_i)^\mathsf{T}\boldsymbol{u} = \boldsymbol{g}_d \tag{16}$$

approximates the boundary condition, where similarly to before, vectors $\boldsymbol{g}_d$ and $\boldsymbol{u}$ represent values of function g_d and unknowns u in stencil nodes of $\boldsymbol{x}_i$. Another set of N_g equations is needed to determine the unknowns introduced by ghost nodes. Additionally to (15) and (16), we also enforce (14) to hold for boundary nodes.

All $N_i + N_d + N_n + N_g$ equations are assembled into a sparse system with $n(N_i + N_n + N_g) + N_d$ non-zero elements in general. The solution u_h of this system is a numerical approximation of u, excluding the values obtained in ghost nodes.

2.4 Note on implementation

We implemented the solution procedure described in this section in C++ using object oriented approach and C++'s strong template system to achieve modularity and consequent dimension independence. The strongest advantage of the presented method is that all building blocks, namely *node positioning*, *stencil selection*, *differential operator approximation* and *PDE discretization*, are independent and can therefore be elegantly coded as abstract modules, not knowing about each other in the core of their implementation. To ease the implementation of the solution procedure, additional abstractions, such as *operators*, *basis functions*, *domain shapes* and *approximations*, are introduced, acting as interfaces between the main blocks. For example, to construct a RBF-FD approximation, one combines the RBF basis class with an augmented RBF-FD class, computes stencil weights and supplies the computed weights into the "operators" class that enables the user to explicitly transform governing equations into the C++ code, as demonstrated in the listing 1.

Vector and scalar fields are implemented as plain arrays, using a well developed linear algebra library [11] that also implements or otherwise supports various direct and iterative linear solvers. Please, refer to our open source Medusa library [29] for more examples and features.

```
// Define differential operator approximation.
Monomials<vec> mon(m);
Polyharmonic<double, k> ph;
RBFFD<decltype(ph), vec, ScaleToFarthest> appr(ph, mon);

// Compute stencil weights (shapes) with RBF-FD.
auto storage = domain.computeShapes<sh::lap|sh::d1>(appr);
Eigen::SparseMatrix<double, Eigen::RowMajor> M(N, N);
M.reserve(storage.supportSizes());
Eigen::VectorXd rhs(N); rhs.setZero();

// Prepare "operators" abstraction.
auto op = storage.implicitOperators(M, rhs);

// PDE discretization.
// Interior.
for (int i : interior) {
    op.lap(i) = f_lap(domain.pos(i));
}
// Dirichlet boundary.
for (int i : dir) {
    op.value(i) = f(domain.pos(i));
    op.lap(i, gh[i]) = f_lap(domain.pos(i));
}
// Neumann boundary.
for (int i : neu) {
    op.neumann(i, domain.normal(i)) = f_grad(domain.pos(i));
    op.lap(i, gh[i]) = f_lap(domain.pos(i));
}
```

Listing 1: A part of dimension-independent source code showing definition and sparse system assembly.

3 Numerical example

The behavior of the proposed solution procedure and its implementation are studied on a Poisson problem with mixed boundary conditions. The aim is to analyze accuracy and convergence properties in one, two and three dimensions. Furthermore, theoretical computational complexity is discussed and supported by experimental measurements of execution time, which allows us to quantify the accuracy vs. execution time trade-off.

The problem is solved on an irregular domain Ω, defined as $\Omega = (B_0 \cup B_1) \setminus (B_2 \cup B_3)$, where

$$B_0 = \left\{ \boldsymbol{x} \in \mathbb{R}^4, \ \left\| \boldsymbol{x} - \frac{\boldsymbol{1}}{\boldsymbol{2}} \right\| < \frac{1}{2} \right\}, \tag{17}$$

$$B_1 = \left\{ \boldsymbol{x} \in \mathbb{R}^4, \ \left\| \boldsymbol{x} - \frac{\boldsymbol{1}}{\boldsymbol{5}} \right\| \leq \frac{1}{4} \right\}, \tag{18}$$

$$B_2 = \left\{ \boldsymbol{x} \in \mathbb{R}^4, \ \left\| \boldsymbol{x} - \frac{\boldsymbol{1}}{\boldsymbol{2}} \right\| \leq \frac{1}{10} \right\} \text{ and} \tag{19}$$

$$B_3 = \left\{ \boldsymbol{x} \in \mathbb{R}^4, \ \| \boldsymbol{x} - \boldsymbol{1} \| \leq \frac{1}{2} \right\} \tag{20}$$

are balls in $\mathbb{R}^d$. For later use, the boundary $\partial\Omega$ is divided into Γ_d and Γ_n, the left and the right half of the boundary, respectively

$$\Gamma_d = \left\{ \boldsymbol{x} \in \partial\Omega, x_1 < \frac{1}{2} \right\}, \tag{21}$$

$$\Gamma_n = \left\{ \boldsymbol{x} \in \partial\Omega, x_1 \geq \frac{1}{2} \right\}. \tag{22}$$

3.1 Governing equation

Numerical solution u_h of Poisson's equation with both Dirichlet and Neumann boundary conditions is studied:

$$\nabla^2 u(\boldsymbol{x}) = f_{lap}(\boldsymbol{x}) \qquad \text{in } \Omega, \tag{23}$$

$$u(\boldsymbol{x}) = f(\boldsymbol{x}) \qquad \text{on } \Gamma_d, \tag{24}$$

$$\nabla u(\boldsymbol{x}) = \boldsymbol{f}_{grad}(\boldsymbol{x}) \qquad \text{on } \Gamma_n. \tag{25}$$

Here, the right hand side was chosen as

$$f(\boldsymbol{x}) = \frac{E(\boldsymbol{x})}{g(\boldsymbol{x})}, \tag{26}$$

where

$$E(\boldsymbol{x}) = \exp\Big(\sum_{i=1}^{d} x_i^{a_i}\Big), \quad g(\boldsymbol{x}) = 1 + \boldsymbol{x}^\mathsf{T}\mathbf{H}\boldsymbol{x}, \quad a_i = 2 + i, \tag{27}$$

$\mathbf{H}$ is a Hilbert matrix of size d, and $\hat{\mathbf{e}}_i$ is the i-th unit vector. The values of Laplacian and the gradient are computed from f as

$$\begin{aligned} f_{lap}(\boldsymbol{x}) = {} & \frac{8E(\boldsymbol{x})}{g(\boldsymbol{x})^3}(\mathbf{H}\boldsymbol{x})^\mathsf{T}(\mathbf{H}\boldsymbol{x}) - \frac{2E(\boldsymbol{x})}{g(\boldsymbol{x})^2}\Big[2(\mathbf{H}\boldsymbol{x})^\mathsf{T}(\sum_{i=1}^{d} a_i x_i^{a_i-1}\hat{\mathbf{e}}_i) + \mathrm{Tr}(\mathbf{H})\Big] \\ & + \frac{E(\boldsymbol{x})}{g(\boldsymbol{x})}\Big[\sum_{i=1}^{d} a_i(a_i-1)x_i^{a_i-2} + (\sum_{i=1}^{d} a_i x_i^{a_i-1}\hat{\mathbf{e}}_i)^\mathsf{T}(\sum_{i=1}^{d} a_i x_i^{a_i-1}\hat{\mathbf{e}}_i)\Big], \end{aligned} \tag{28}$$

$$\boldsymbol{f}_{grad}(\boldsymbol{x}) = \frac{E(\boldsymbol{x})}{g(\boldsymbol{x})}\Big[\sum_{i=1}^{d} a_i x_i^{a_i-1}\hat{\mathbf{e}}_i - \frac{2}{g(\boldsymbol{x})}(\mathbf{H}\boldsymbol{x})^\mathsf{T}\Big]. \tag{29}$$

The closed-form solution f of the above problem is a rational non-easily separable function allowing us to validate the numerically obtained solution u_h. The computed u_h is only known at discretization points $\boldsymbol{x}_i$. The errors between u_h and u are measured in three different norms:

$$e_1 = \frac{\|u_h - u\|_1}{\|u\|_1}, \quad \|u\|_1 = \frac{1}{N}\sum_{i=1}^{N}|u_i|, \tag{30}$$

$$e_2 = \frac{\|u_h - u\|_2}{\|u\|_2}, \quad \|u\|_2 = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|u_i|^2}, \tag{31}$$

$$e_\infty = \frac{\|u_h - u\|_\infty}{\|u\|_\infty}, \quad \|u\|_\infty = \max_{i=1,\dots,N}|u_i|. \tag{32}$$

The problem (23–25) is studied in $d \in \{1, 2, 3\}$ dimensions. Scattered computational nodes are generated using a dimension-agnostic node positioning algorithm described in section 2.1. Ghost nodes were added to both Dirichlet and Neumann boundaries, and are excluded from any post-processing. An example of node distribution is shown in figure 2.

Numerical results are computed using RBF-FD with PHS radial basis function $\phi(r) = r^3$ and monomial augmentation, as described in section 2. Radial function was kept same for all cases; however, various orders of monomial augmentation were tested. For each dimension d, solution to the problem is obtained using monomials up to and including degree m, for $m \in \{-1, 0, 2, 4, 6, 8\}$, where $m = -1$ represents a pure RBF case with no monomials added. Only even orders of m were used, because the same order of convergence is observed with odd powers, but at a higher computational cost [6].

Stencils for each node were selected by taking the closest n nodes, where n was equal to two times the number of augmenting monomials, as recommended by Bayona [3], or at least a FDM minimum of $2d + 1$, i.e.

$$n = \max\left\{2\binom{m+d}{d},\ 2d+1\right\}. \tag{33}$$

m	$d=1$	$d=2$	$d=3$
-1	3	5	7
0	3	5	7
2	6	12	20
4	10	30	70
6	14	56	168
8	18	90	330

Table 1 Support sizes in different dimensions for various augmentation orders.

Specific values for m, n and d are presented in table 1.

BiCGSTAB with ILUT preconditioner was used to solve the sparse system. Global tolerance was set to 10^{-15} with a maximum number of 500 iterations, while the drop tolerance and fill-factor were dimension dependent: 10^{-4} and 20 for $d = 1$, 10^{-4} and 30 for $d = 2$, and 10^{-5} and 50 for $d = 3$, respectively.

Figure 2 shows three examples of computed numerical solution u_h for each domain dimension d. The solutions are shown for various values of m and for small enough values of N to also show nodal distributions.

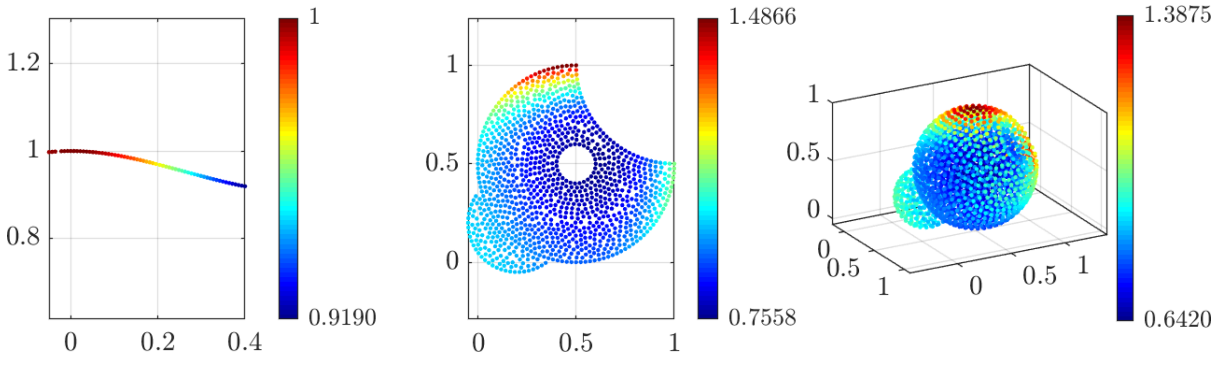

Fig. 2 Computed numerical solution u_h for $d = 1, 2, 3$, from left to right. Chosen highest polynomial degree m and node count N are as follows: $N = 64$ and $m = 4$ for $d = 1$, $N = 1286$ and $m = 2$ for $d = 2$ and $N = 3850$ and $m = 4$ for $d = 3$.

In the top row of figure 3 global sparse matrices are shown. Additionally, spectra of the Laplacian differentiation matrices for cases shown in figure 2 are shown in the bottom row of figure 3, to better assess the approximation quality. For all three cases, the eigenvalues have negative real parts with relatively small spread around the imaginary axis.

3.2 Convergence rate

When using RBF-FD augmented with monomials, consistency is ensured up to order m, which makes the expected convergence rate of at least $O(h^m)$. Here, h denotes the nodal spacing, which is inversely proportional to $\sqrt[d]{N}$.

Figure 4 shows e_1, e_2 and e_∞ errors for various augmentation orders in two dimensions. The three errors have very similar values and similar convergence rates. Convergence rates were estimated by computing the slope of a least-squares linear trend line over the appropriate subset of the data. Divergence is observed in the $m = 0$ and the $m = -1$ case, which is consistent with properties of PHS RBFs. These two cases are excluded from any further analyses in this paper.

In the rest of the discussion, only e_∞ is used for convergence analysis, since it measures the lowest convergence rates and does not involve averaging, contrary to e_1 and e_2.

Figure 5 shows the e_∞ error for $d = 1$, $d = 2$, and $d = 3$ dimensions. The span of the horizontal axis was chosen in such a way that the total number of nodes in the largest case was around $N = 10^5$

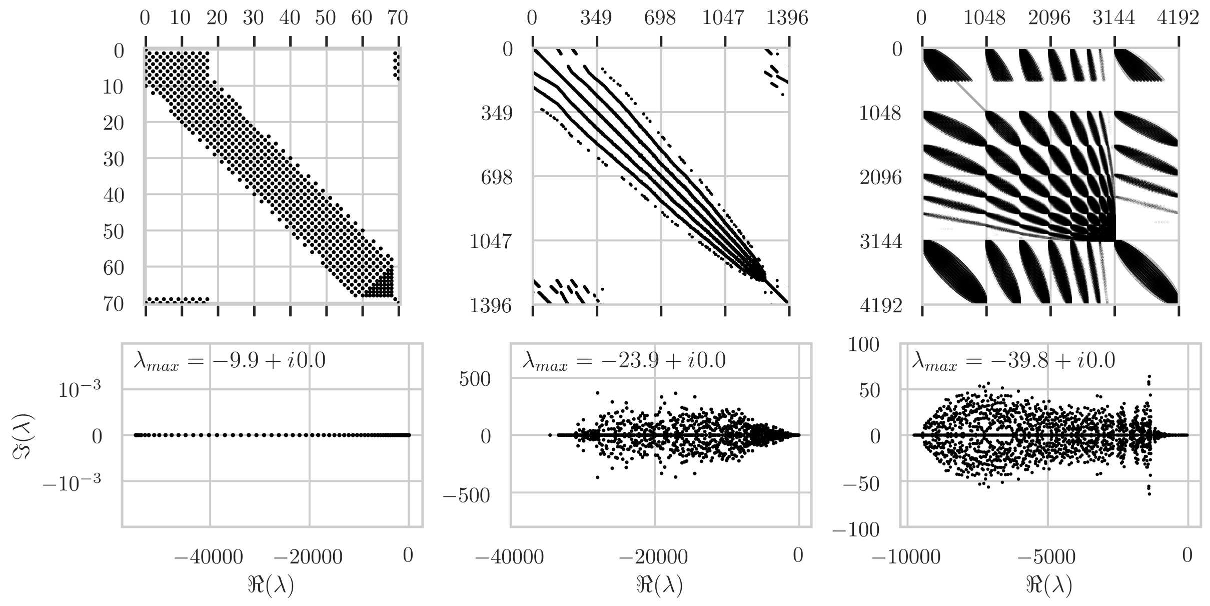

Fig. 3 Plots of global sparse matrices (top row) and spectra of the Laplacian differentiation matrices (bottom row), corresponding to the solutions in figure 2.

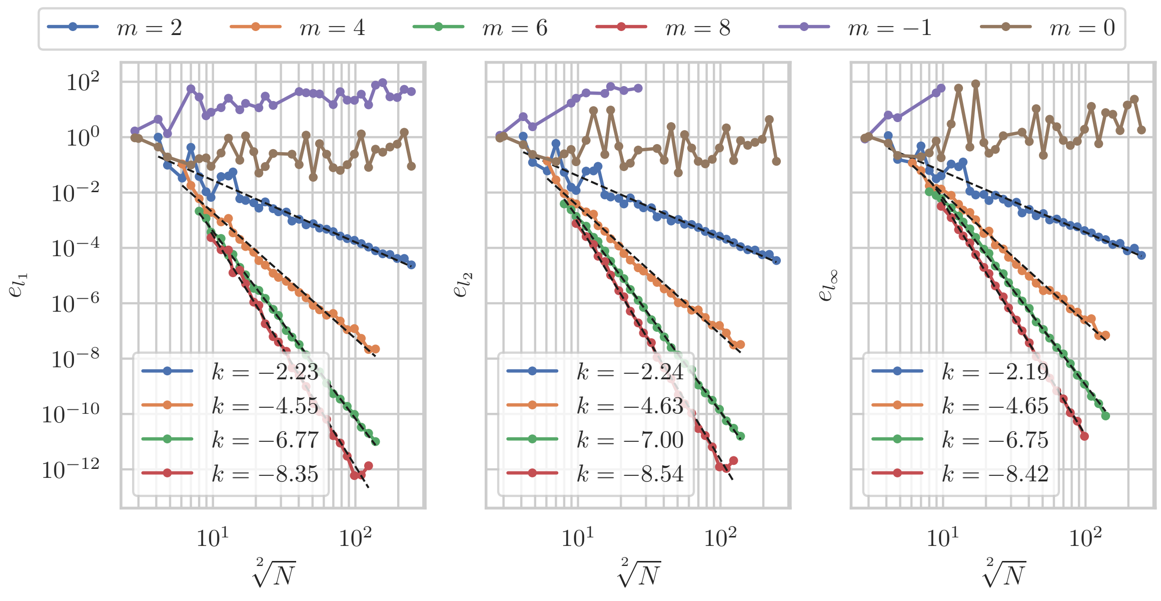

Fig. 4 Errors between analytical solution u and numerically obtained u_h, measured in three different norms. Computed are e_1, e_2 and e_∞, from left to right, respectively, for the $d = 2$ dimensional case.

in all dimensions. The observed convergence rates are independent of domain dimension and match the predicted order $O(h^m)$.

All of the plots in the $d = 1$ case eventually diverge, due to the errors in finite precision arithmetic, as previously noted for interpolation by Flyer et al. [6]. The dotted line in the $d = 1$ case shows the ε/h^2 line, where $\varepsilon \approx 2.22 \cdot 10^{-16}$. The numerically obtained solution for the $d = 3$ and $m = 8$ case is unstable for smaller N. For higher node counts N, the expected convergence behavior is obtained, as seen from the fitted dashed line.

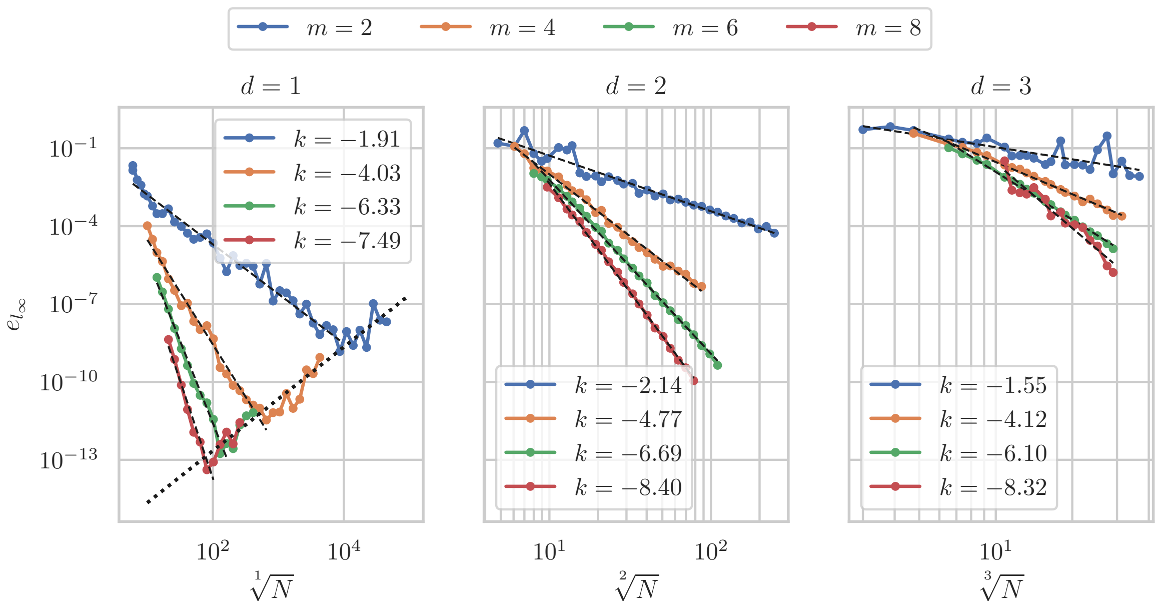

Fig. 5 Convergence rate of e_∞ for all domain dimensions $d = 1, 2, 3$, from left to right, respectively.

3.3 Computational efficiency

The importance of several different stages of u_h computation is studied. The computational procedure is divided into

- *node positioning*, where quasi-uniform placing of nodes in the domain Ω and the domain boundary $\partial\Omega$, including positioning of N_g ghost nodes, takes place. Node positioning time also includes finding the stencils for each node in the domain,
- *stencil weights computation*, where basis functions are defined and shapes for the Laplace operator and first derivatives are stored,
- *system assembly*, where computed weights are assembled in a sparse matrix and its right-hand side is computed and
- *system solution*, where the sparse system is solved.

3.3.1 Computational complexity

The theoretical computational complexity is analyzed in this section. The total number of nodes will be denoted as $N_t = N + N_g$; however, as N_g nodes are distributed only along the boundary, it holds that $N_g = O(N^{\frac{d-1}{d}})$ and thus $N_t = O(N)$.

The node positioning algorithm has complexity $O(N_t \log N_t)$ [30]. Finding stencils of n closest nodes takes $O(nN_t \log N_t)$ time, using a fast spatial search structure, such as a k-d tree. The computation of stencils weights performs N_t solutions of linear systems of size $(n+s) \times (n+s)$, where $s = \binom{m+d}{d}$ is the number of monomials used for augmentation. Since n was chosen to be at least $2s$, it holds that $s = O(n)$. Using LU decomposition or any other standard solution procedure for dense linear systems takes $O((n+s)^3) = O(n^3)$ time. The total cost of weight computation is therefore $O(n^3 N_t)$.

With appropriate pre-allocation of storage for the sparse matrix, system assembly takes linear time in number of stencil nodes for each node, and right hand-side computation taken $O(1)$ per node. The total cost of system assembly is thus $O(nN_t)$.

The solution of the sparse system uses iterative BiCGSTAB with ILUT preconditioner, whose speed of convergence depends on the matrix properties.

The time complexity of the complete procedure is

$$O(nN_t \log N_t + n^3 N_t) + T,$$

where T is the complexity of the sparse solver.

3.3.2 Execution time

In this section, we measure execution time spent on different parts of the solution procedure. All computations were performed on a single core of a computer with `Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz` processor and 64 GB of DDR4 memory. Code was compiled using `g++ (GCC) 8.1.0` for Linux with `-O3 -DNDEBUG` flags.

Total execution times are shown in figure 6 and correspond to accuracy results in figure 5. The computational time grows with N and with m, as expected from theoretical predictions in section 3.3.1.

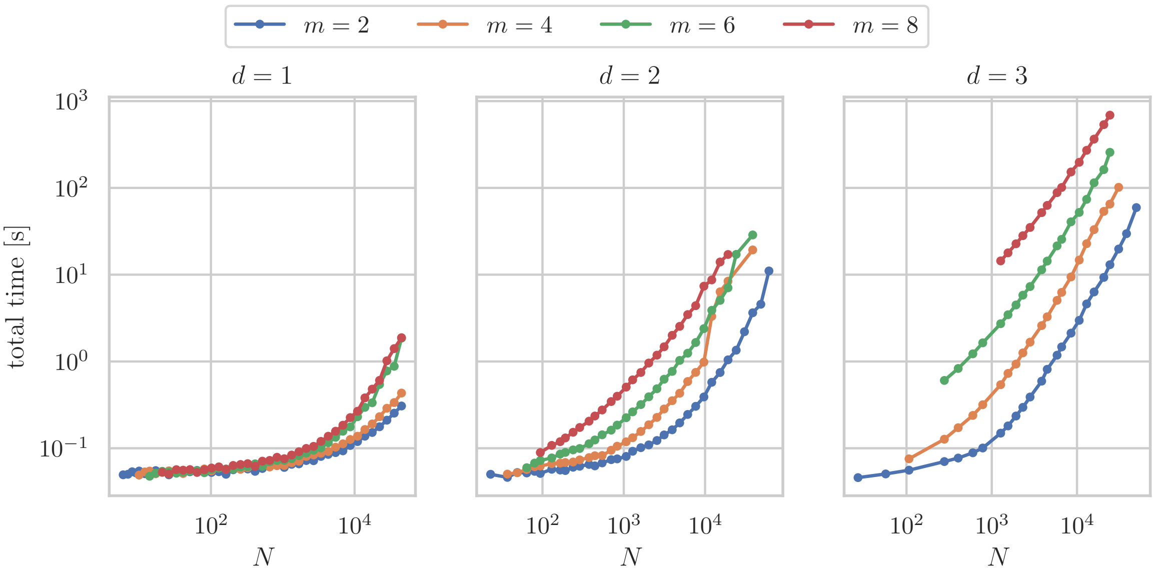

Fig. 6 Median of 10 total execution times of u_h computation for various setups.

Absolute times of different computation stages and their proportions to the total time are shown in figure 7, on the left and the right side, respectively. The observed growth rates match the theoretical complexities predicted for node positioning, weight computation and system assembly.

Relative execution times provide additional insight into the execution of the solution procedure and into optimization and parallelization opportunities. The majority of the computational time is usually spent on either computing the stencil weights (for smaller N) or on system solution (for large N). Similar behavior was observed for other m and in other dimensions, with different percentage of total time spent on node positioning, weight computation and system solution [15].

3.4 Accuracy vs. execution time

In the previous sections, we have shown that using higher orders, both accuracy and execution time increase. In this section, we analyze the accuracy vs. execution time trade-off. Figure 8 shows e_∞ error plotted with respect to the total computational time needed to achieve it.

Significant differences can be observed between different orders of monomial augmentation. For prototyping or any other sort of quick scanning of how or if the computed solution u_h converges, using polynomials of a lower degree is undeniably very beneficial – the computation of u_h takes little time, but at a cost of limited accuracy. When higher accuracy is required, using polynomials of a higher degree can lead to a several orders faster computation time. In some cases, using higher orders might even be a necessity, e.g. for $d = 2$, where accuracy of $e_\infty \approx 10^{-10}$ is reached the fastest by $m = 8$, while solution for $m = 2$ would require N out of reasonable computing capabilities. The findings are summarized in table 2.

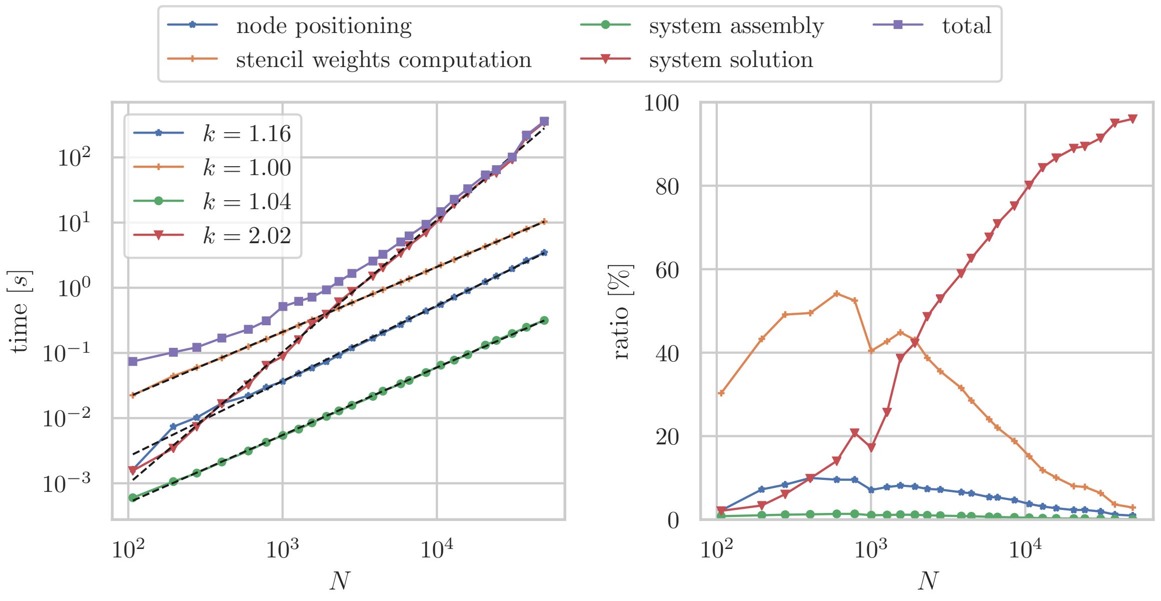

Fig. 7 Absolute and relative times of different parts of the solution procedure for $d = 3$ and $m = 4$.

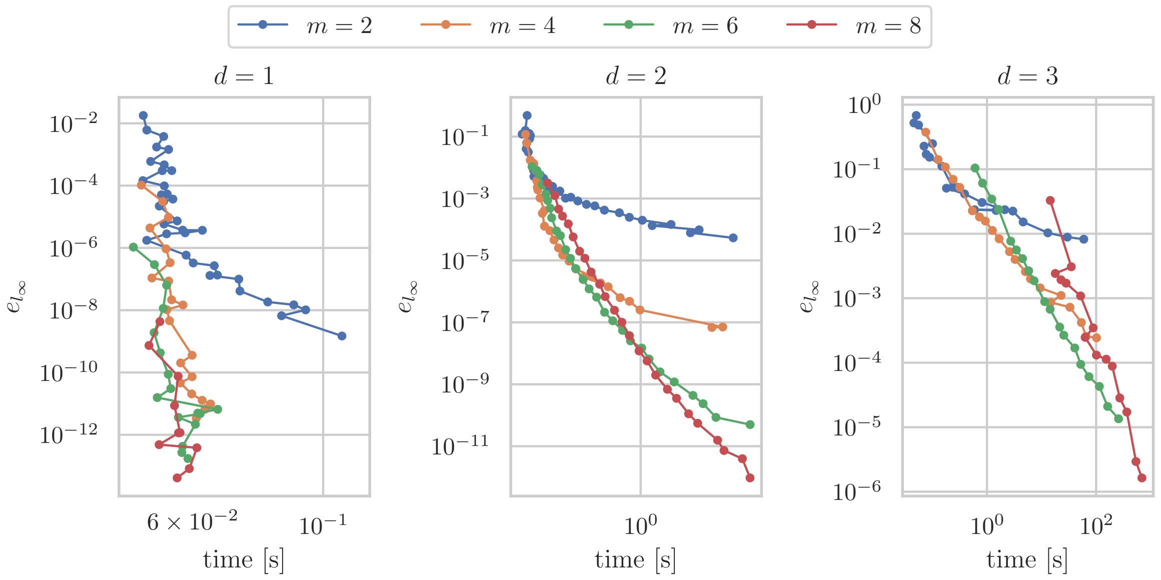

Fig. 8 Accuracy vs. execution time trade-off for different orders of monomial augmentation.

$d = 1$		$d = 2$		$d = 3$	
target accuracy e_∞	optimal m	target accuracy e_∞	optimal m	target accuracy e_∞	optimal m
10^0 to 10^{-4}	2	10^0 to 10^{-2}	2	10^0 to 10^{-1}	2
10^{-4} to 10^{-6}	4	10^{-2} to 10^{-5}	4	10^{-1} to 10^{-3}	4
10^{-6} to 10^{-8}	6	10^{-5} to 10^{-8}	6	10^{-3} to 10^{-5}	6
10^{-8} to 10^{-13}	8	10^{-8} to 10^{-12}	8	10^{-5} to 10^{-7}	8

Table 2 Optimal setups for various desired target accuracy ranges in 1, 2 and 3 dimensions.

Using the data in the table, we can extract a rough general recommendation. As a rule of thumb, for the desired accuracy $e_\infty = 10^{-k}$ and dimension d, the recommended order of augmentation is

$$m = \frac{5}{4}k + \frac{4}{5}d - 2, \tag{34}$$

rounded to the nearest positive even integer. Even though the data points in the table are close to being planar, the formula (34) does not necessarily generalize well. A more general rule is that the order of monomials should be increased with every two to three orders of increase in accuracy, and that higher order augmentation should be more aggressively used in higher dimensions.

4 Additional example

In addition to already solved cases, we now demonstrate a solution of a 4-dimensional Poisson problem (23–25). The irregular domain Ω is now defined as $\Omega = B_0 \setminus (B_1 \cup B_2 \cup B_3)$, where

$$B_0 = \left\{ \boldsymbol{x} \in \mathbb{R}^4,\ \left\| \boldsymbol{x} - \frac{\mathbf{1}}{\mathbf{2}} \right\| < \frac{1}{2} \right\}, \tag{35}$$

$$B_1 = \left\{ \boldsymbol{x} \in \mathbb{R}^4,\ \left\| \boldsymbol{x} - \left(\frac{1}{2}, 1, \frac{1}{2}, \frac{1}{2} \right) \right\| \le \frac{1}{4} \right\}, \tag{36}$$

$$B_2 = \left\{ \boldsymbol{x} \in \mathbb{R}^4,\ \| \boldsymbol{x} - \mathbf{0} \| \le \frac{13}{16} \right\} \text{ and} \tag{37}$$

$$B_3 = \left\{ \boldsymbol{x} \in \mathbb{R}^4,\ \left\| \boldsymbol{x} - \left(\frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{1}{2} \right) \right\| \le \frac{1}{8} \right\} \tag{38}$$

are balls in $\mathbb{R}^4$.

Dirichlet and Neumann boundary conditions are defined similarly to before, i.e., Γ_d is the left half and Γ_n is the right half of $\partial\Omega$. Additionally, the boundary of the smallest ball ∂B_3 is added to the Dirichlet boundary:

$$\Gamma_d = \left\{ \boldsymbol{x} \in \partial\Omega, x_1 < \frac{1}{2} \right\} \cup \partial B_3, \tag{39}$$

$$\Gamma_n = \left\{ \boldsymbol{x} \in \partial\Omega, x_1 \ge \frac{1}{2} \right\} \setminus \partial B_3. \tag{40}$$

Scattered computational nodes were positioned using the same dimension-agnostic node positioning algorithm as before. A numerical solution u_h was obtained using RBF-FD with PHS $\phi(r) = r^3$ augmented with polynomials of degree $m = 4$, according to our rule of thumb (34) for the desired accuracy $e_\infty = 10^{-2}$.

Approximately $N = 85000$ nodes were positioned in Ω and closest $n = 950$ nodes were selected as stencils for each node from the domain. Ghost nodes were, as in the previous case, added to both Dirichlet and Neumann boundaries, and excluded from any post-processing. The final system was solved using a direct sparse solver.

Figure 9 shows the numerically obtained solutions. Four three-dimensional slices are shown, defined by setting one coordinate to $x_i = 1/2$. Modified Sheppard's interpolation algorithm [10] was used to interpolate the solution to an intermediate grid, used for plotting the slices.

The solution is well-behaved even in 4 dimensions; however, a relatively large support size is needed to obtain a desirable numerical stability. The errors equal to $e_1 = 6.83 \cdot 10^{-4}$, $e_2 = 2.11 \cdot 10^{-3}$ and $e_\infty = 1.72 \cdot 10^{-2}$. The total computational time spent was approximately 15 hours.

5 Conclusions

The message of this paper is twofold. First, we demonstrated that it is possible to design an appropriately abstract implementation, which encompasses most of the meshless mathematical elegance, allowing the user to construct a high order dimension-independent solution procedure. To fully demonstrate the dimensional independence, we also presented a solution of a 4-dimensional Poisson's problem on an irregular domain with both Neumann and Dirichlet boundary conditions.

Second, we used the devised implementation to analyze the increasing execution time that comes tied with high order augmentation, to determine the conditions of optimal computation efficiency for a desired target accuracy.

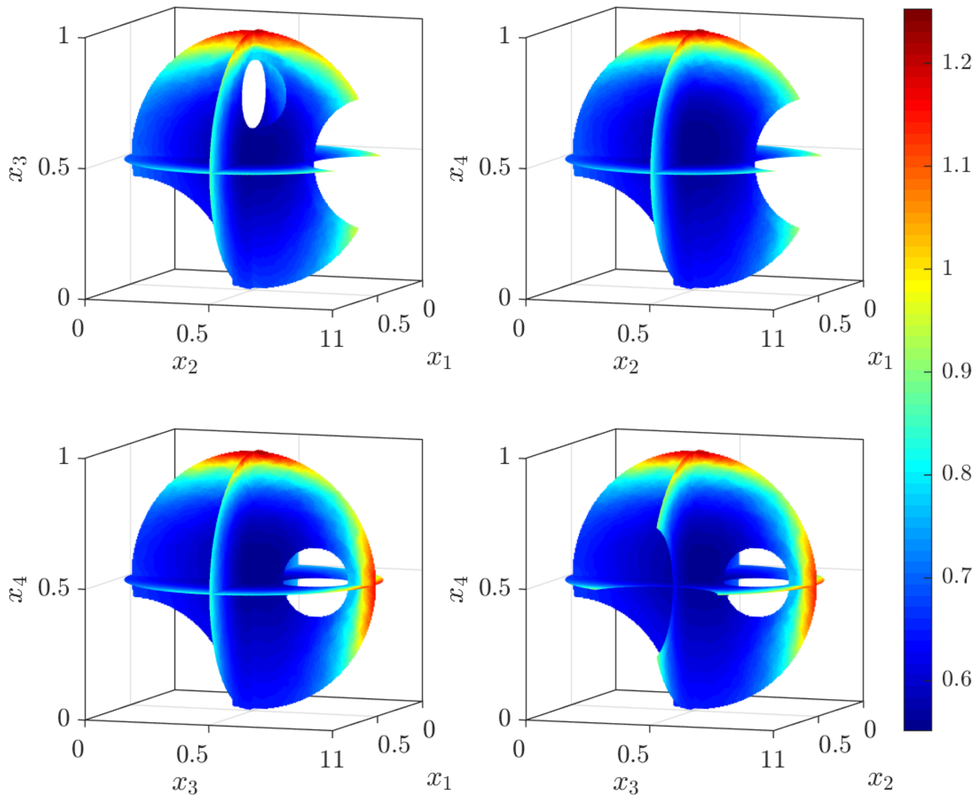

Fig. 9 3-dimensional cross sections of a solution to a 4-dimensional Poisson problem.

The analyses are performed on the solution of a Poisson problem with mixed boundary conditions in one, two and three dimensions. To avoid shape parameter dependency, we used PHS augmented with monomials as RBFs. Scattered nodes were positioned with a dedicated dimension-agnostic node generation algorithm. The theoretical findings on how the highest order of the augmenting polynomial directly controls the approximation rate of the RBF-FD independently of the domain dimension are verified. A detailed breakdown of the computational complexity and the execution time of different computational stages is also provided, to ensure that the implementation agrees with the theoretical predictions. Finally, the high order vs. execution time trade-off is analyzed and the findings are summarized in figure 8 and table 2. While the analyses were done only for this particular problem, the results can be generalized in the sense that for a high target accuracy, a high order method is a better choice, and vice versa.

Another interesting point are the increasing stencil sizes required for high order methods, as shown in table 1. Especially in higher dimensions, this cost quickly becomes unmanageable. Therefore, our future work will be focused primarily on better understanding of the impact of the stencil size on the approximation quality.

Acknowledgements

The authors would like to acknowledge the financial support of the Slovenian Research Agency (ARRS) research core funding No. P2-0095 and the Young Researcher program PR-08346.

References

1. Ahmad, I., Islam, S.u.I., Khaliq, A.Q.: Local RBF method for multi-dimensional partial differential equations. Computers & Mathematics with Applications **74**(2), 292–324 (2017). DOI 10.1016/j.camwa.2017.04.026
2. Bayona, V.: An insight into RBF-FD approximations augmented with polynomials. Computers & Mathematics with Applications **77**(9), 2337–2353 (2019). DOI 10.1016/j.camwa.2018.12.029
3. Bayona, V., Flyer, N., Fornberg, B., Barnett, G.A.: On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs. Journal of Computational Physics **332**, 257–273 (2017). DOI 10.1016/j.jcp.2016.12.008

4. Bridson, R.: Fast Poisson disk sampling in arbitrary dimensions. In: SIGGRAPH sketches, p. 22 (2007). DOI 10.1145/1278780.1278807
5. Choi, Y., Kim, S.: Node generation scheme for meshfree method by Voronoi diagram and weighted bubble packing. In: Fifth us national congress on computational mechanics, Boulder, CO (1999)
6. Flyer, N., Fornberg, B., Bayona, V., Barnett, G.A.: On the role of polynomials in RBF-FD approximations: I. Interpolation and accuracy. Journal of Computational Physics **321**, 21–38 (2016). DOI 10.1016/j.jcp.2016.05.026
7. Fornberg, B., Flyer, N.: Fast generation of 2-D node distributions for mesh-free PDE discretizations. Computers & Mathematics with Applications **69**(7), 531–544 (2015)
8. Fornberg, B., Flyer, N.: A primer on radial basis functions with applications to the geosciences, *CBMS-NSF Regional Conference Series in Applied Mathematics*, vol. 87. SIAM (2015). DOI 10.1137/1.9781611974041
9. Fornberg, B., Flyer, N.: Solving PDEs with radial basis functions. Acta Numerica **24**, 215–258 (2015). DOI 10.1017/S0962492914000130
10. Franke, R., Nielson, G.: Smooth interpolation of large sets of scattered data. International journal for numerical methods in engineering **15**(11), 1691–1704 (1980). DOI 10.1002/nme.1620151110
11. Guennebaud, G., Jacob, B., et al.: Eigen v3. `http://eigen.tuxfamily.org` (2010)
12. Hardin, D., Saff, E.: Discretizing manifolds via minimum energy points. Notices of the AMS **51**(10), 1186–1194 (2004)
13. Jančič, M., Slak, J., Kosec, G.: Standalone implementation of solution to the Poisson's equation. `http://e6.ijs.si/medusa/static/DimensionIndependentPoisson.zip` (2019)
14. Kosec, G.: A local numerical solution of a fluid-flow problem on an irregular domain. Advances in engineering software **120**, 36–44 (2018). DOI 10.1016/j.advengsoft.2016.05.010
15. Kosec, G., Slak, J.: Parallel RBF-FD solution of the Boussinesq's problem. In: P. Iványi, B.H.V. Topping (eds.) Proceedings of the Sixth International Conference on Parallel, Distributed, GPU and Cloud Computing for Engineering, June 5–6, 2019, Pécs, Hungary, Civil-comp proceedings. Stirlingshire: Civil-Comp Press (2019)
16. Liu, G.R.: Mesh free methods: moving beyond the finite element method, first edn. CRC press, Boca Raton (2002). DOI 10.1201/9781420040586
17. Liu, Y., Nie, Y., Zhang, W., Wang, L.: Node placement method by bubble simulation and its application. Computer Modeling in Engineering and Sciences (CMES) **55**(1), 89 (2010)
18. Löhner, R., Oñate, E.: A general advancing front technique for filling space with arbitrary objects. International journal for numerical methods in engineering **61**(12), 1977–1991 (2004)
19. Mairhuber, J.C.: On Haar's theorem concerning Chebychev approximation problems having unique solutions. Proceedings of the American Mathematical Society **7**(4), 609–615 (1956). DOI 10.2307/2033359
20. Maksić, M., Djurica, V., Souvent, A., Slak, J., Depolli, M., Kosec, G.: Cooling of overhead power lines due to the natural convection. International Journal of Electrical Power & Energy Systems **113**, 333–343 (2019). DOI 10.1016/j.ijepes.2019.05.005
21. Mavrič, B., Šarler, B.: Equivalent-PDE based stabilization of strong-form meshless methods applied to advection-dominated problems. Engineering Analysis with Boundary Elements **113**, 315–327 (2020). DOI 10.1016/j.enganabound.2020.01.014
22. Milovanović, S., von Sydow, L.: Radial basis function generated finite differences for option pricing problems. Computers & Mathematics with Applications **75**(4), 1462–1481 (2018). DOI 10.1016/j.camwa.2017.11.015
23. Oanh, D.T., Davydov, O., Phu, H.X.: Adaptive RBF-FD method for elliptic problems with point singularities in 2D. Applied Mathematics and Computation **313**, 474–497 (2017). DOI 10.1016/j.amc.2017.06.006
24. Onate, E., Idelsohn, S., Zienkiewicz, O.C., Taylor, R.L.: A finite point method in computational mechanics. Applications to convective transport and fluid flow. International journal for numerical methods in engineering **39**(22), 3839–3866 (1996). DOI 10.1002/(sici)1097-0207(19961130)39:22<3839::aid-nme27>3.0.co;2-r
25. Shankar, V., Fogelson, A.L.: Hyperviscosity-based stabilization for radial basis function-finite difference (RBF-FD) discretizations of advection–diffusion equations. Journal of computational physics **372**, 616–639 (2018). DOI 10.1016/j.jcp.2018.06.036
26. Shankar, V., Kirby, R.M., Fogelson, A.L.: Robust node generation for meshfree discretizations on irregular domains and surfaces. SIAM Journal on Scientific Computing **40**(4), 2584–2608 (2018). DOI 10.1137/17m114090x
27. Slak, J., Kosec, G.: Standalone implementation of the proposed node placing algorithm (2018). `http://e6.ijs.si/medusa/static/PNP.zip`
28. Slak, J., Kosec, G.: Adaptive radial basis function–generated finite differences method for contact problems. International Journal for Numerical Methods in Engineering **119**(7), 661–686 (2019). DOI 10.1002/nme.6067
29. Slak, J., Kosec, G.: Medusa: A C++ library for solving pdes using strong form mesh-free methods (2019). URL `http://e6.ijs.si/medusa/`
30. Slak, J., Kosec, G.: On generation of node distributions for meshless PDE discretizations. SIAM Journal on Scientific Computing **41**(5), A3202–A3229 (2019). DOI 10.1137/18M1231456
31. Slak, J., Kosec, G.: Refined meshless local strong form solution of Cauchy–Navier equation on an irregular domain. Engineering analysis with boundary elements **100**, 3–13 (2019). DOI 10.1016/j.enganabound.2018.01.001
32. Tolstykh, A.I., Shirobokov, D.A.: On using radial basis functions in a "finite difference mode" with applications to elasticity problems. Computational Mechanics **33**(1), 68–79 (2003). DOI 10.1007/s00466-003-0501-9
33. Wendland, H.: Scattered data approximation, *Cambridge Monographs on Applied and Computational Mathematics*, vol. 17. Cambridge university press (2004). DOI 10.1017/cbo9780511617539
34. Wright, G.B., Fornberg, B.: Stable computations with flat radial basis functions using vector-valued rational approximations. Journal of Computational Physics **331**, 137–156 (2017). DOI 10.1016/j.jcp.2016.11.030
35. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '93, pp. 311–321. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1993). URL `http://dl.acm.org/citation.cfm?id=313559.313789`