Journal of Peridynamics and Nonlocal Modeling (2024) 6:87–117
https://doi.org/10.1007/s42102-023-00108-6

RESEARCH

nlfem: a Flexible 2d FEM Python Code for Nonlocal Convection-Diffusion and Mechanics

Manuel Klar[1] · Christian Vollmann[1] · Volker Schulz[1]

Received: 12 August 2022 / Accepted: 31 August 2023 / Published online: 26 September 2023

Abstract
In this work, we present the mathematical foundation of an assembly code for finite element approximations of nonlocal models with compactly supported, weakly singular kernels. We demonstrate the code on a nonlocal diffusion model in various configurations and on a two-dimensional bond-based peridynamics model. Further examples can be found in D'Elia et al. (Math Models Methods Appl Sci 31(08):1505–1567, 2021). The code nlfem is published under the GNU General Public License (for details, see, e.g., https://www.gnu.org/licenses/gpl-3.0.de.html) and can be freely downloaded at https://gitlab.uni-trier.de/pde-opt/nonlocal-models/nlfem.

Keywords Nonlocal operators · Finite element discretizations · Python

1 Introduction

Differential equations yield solutions which necessarily contain a certain amount of regularity and are based on local interactions. There are many real-world phenomena where those inherent assumptions are violated. Therefore, over the last two decades, nonlocal models attracted attention due to their capability of circumventing those limitations [2].

The models emerge due to various applications like anomalous diffusion [3, 4], peridynamics [5], or image processing [6, 7] and are diverse in their mathematical nature [8, 9]. That is, the integral kernels might exhibit strong singularities, an infinite or finite interaction horizon, and can be scalar or tensor-valued. Their investigation is often accompanied with numerical experiments motivated by these applications.

In this work, we describe the discretization of nonlocal operators of the general form

✉ Manuel Klar
klar@uni-trier.de

Christian Vollmann
vollmann@uni-trier.de

Volker Schulz
volker.schulz@uni-trier.de

[1] Universitaet Trier, D-54286, Trier, Germany

$$-\mathcal{L}_\delta \mathbf{u}(\mathbf{x}) := 2\int_{\widetilde{\Omega}} \big(\mathbf{C}_\delta(\mathbf{x},\mathbf{y})\mathbf{u}(\mathbf{x}) - \mathbf{C}_\delta(\mathbf{y},\mathbf{x})\mathbf{u}(\mathbf{y})\big)d\mathbf{y},$$

where the kernel $\mathbf{C}_\delta$ vanishes for points farther apart than some horizon $\delta > 0$. The support of the kernel, also referred to as interaction neighborhood in the following, is often modeled by the Euclidean norm ball or suitable approximations thereof.

The purpose of our code nlfem [10] is to compute numerical solutions of related boundary value problems at a convenient speed for researchers. The discretization of these problems is achieved by a finite element approximation. The resulting variational framework comes at the price of a second integration compared to the operator in its strong form, which makes it more costly compared to classical differential operators. Further challenges in the implementation arise due to finite interaction horizons and singularities of the kernel [1].

The finite element method is one among several methods to approximate nonlocal operators. Mesh-free methods [11–13] are commonly applied to peridynamics problems, and there exist finite difference schemes [14] and kernel collocation methods [15] for nonlocal diffusion and mechanics.

As nlfem is a finite element implementation, we shortly review the available codes for nonlocal problems based on that method. The foundation for operators related to the fractional Laplacian[1] is given by boundary element methods [16]. Based on these fundamentals, a MATLAB implementation for a finite element approximation of the two-dimensional fractional Laplacian with infinite interaction is presented in [17]. Further advanced techniques to efficiently implement the fractional Laplacian are developed in [18] and incorporated into the finite element code PyNucleus [19]. The package is a recommendable alternative to our code, and we give a detailed comparison of PyNucleus and nlfem in Sect. 4.3. Apart from that, there exists a commercial finite element code as part of LS-DYNA [20] for peridynamics. For a general overview, we refer the reader to the comprehensive review paper [21] on numerical methods for nonlocal problems.

Our code nlfem assembles nonlocal operators on triangular meshes based on linear continuous Galerkin (CG) or discontinuous Galerkin (DG) **ansatz spaces**.

It allows the assembly of stiffness matrices related problems with given Neumann and Dirichlet **boundary data** [22–24]. An important detail is that the null space of systems corresponding to pure Neumann boundary data is exact due to a careful approximation of kernel truncations which eliminates distortions rooted in geometric errors. The knowledge of the null space can be exploited to efficiently evaluate the pseudoinverse, for example, by rank corrections or Krylov subspace solvers.

Concerning the **domain**, nlfem covers a variety of different configurations. It handles nonlocal interactions in nonconvex or even disconnected domains where the intersection between the interaction neighborhood $B_\delta(\mathbf{x})$ and the domain can be disconnected. For example, this is of particular interest in shape optimization with nonlocal operators [25, 26], where the domain is modified iteratively.

The **kernel** can be symmetric or nonsymmetric as well as scalar—or matrix-valued. For the symmetric case, our discretization of the weak form guarantees the symmetry of the stiffness matrix up to machine precision. The code can generically handle smooth kernels, and it comes with quadrature rules for fractional-type kernels as they are found

[1] Throughout this work, we always refer to the fractional Laplacian in the integral form.

in [16, 17]. In addition to that, the kernel can vary depending on the subdomain it is evaluated on. This opens the door to the assembly of **interface problems** determined by spatially variable kernels.

The nlfem code is most efficient for operators with **interaction horizons** which are comparable to the mesh size, i.e., $h \leqslant \delta \leqslant Ch$ for some $C \geqslant 1$. This relation is often used in the nonlocal mechanics setting; for example, in [27, 28], choices such as $\delta = 3h$ or $\delta = 4h$ are used. A careful consideration of the quadrature and interpolation errors can allow smaller ratios δ/h and increase the sparsity of the related systems.

Our implementation is based on the extensive discussions on the errors incurred by various **approximations of the interaction neighborhood** and quadrature rules [1, 26], all of which are implemented here. We only discuss a small selection of interaction neighborhoods in this paper, such as two approximations of the Euclidean norm ball, which provably do not deteriorate the finite element interpolation error [1]. Furthermore, the interaction neighborhood of a kernel is efficiently determined by a breadth-first traversal [29, Chapter 6] of finite elements throughout the assembly process, which avoids expensive preprocessing computations.

A fundamental advantage of finite element methods is that they can be considered to be **asymptotically compatible** in the sense of [30]. Our code reproduces this property for the cases $h \leqslant \delta$, $h \sim \delta \to 0$ if the implemented interaction neighborhood does not induce geometric errors. For example, this is the case for the implemented infinity norm ball.

For convenience, the assembly is performed in multiple threads, and the main routine, which is written in **C++**, comes with a user-friendly **Python** interface. When it comes to solving, we note that the stiffness matrix is returned in compressed sparse row (CSR) format. Therefore, the user can apply any sparse solver accessible from Python and apply it to the stiffness matrix.

The remainder of this article is organized into two main sections. First, in Sect. 2, we give a precise formulation of the targeted problem class (Sect. 2.1) along with its finite element approximation (Section 2.2). In the following subsections, we then highlight discretization details that deserve special attention in a nonlocal framework. Second, in Sect. 3, we present various numerical examples, including diffusion and mechanics, and give a brief scaling study.

2 Finite Element Approximation

In this section, we review known results about the assembly of finite element approximations to nonlocal operators [1]. However, the current literature does not discuss in detail the influence of ball approximations on the symmetry of the system matrix. That is, a symmetric kernel might be approximated by a nonsymmetric stiffness matrix if the kernel truncation is not handled with sufficient care. The novel contribution in this section is a ball approximation which rules out geometric and quadrature-related distortions. Moreover, we present a novel assembly algorithm tailored to truncated kernels which identifies the interaction neighborhood on-the-fly without the requirement of a prepossessing step.

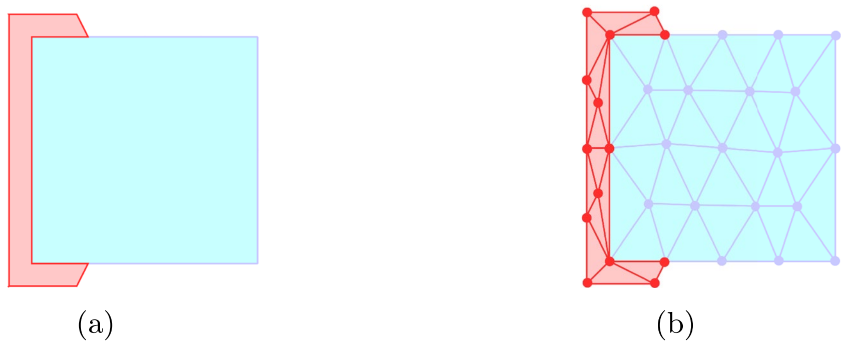

Fig. 1 **a** The compact domain $\widetilde{\Omega}$ contains Ω (blue) and Ω_D (red), where the latter is possibly empty. The blue boundary belongs to Ω, and the red boundary belongs to Ω_D. **b** The red elements belong to the nonlocal Dirichlet boundary Ω_D, while the blue elements belong to the domain Ω. Note that the vertices on the red lines do not belong to the interior of Ω

2.1 Problem Formulation

Let $\widetilde{\Omega} \subset \mathbb{R}^d$ be a compact *domain*, and let $\Omega \subset \widetilde{\Omega}$ be open in $\widetilde{\Omega}$.[2] We note that the case $\Omega = \widetilde{\Omega}$ is allowable, and Ω is open in $\mathbb{R}^d$ only if $\Omega \subset \operatorname{int}(\widetilde{\Omega})$. We refer to Ω as *domain*. The complement of Ω in $\widetilde{\Omega}$ is denoted by $\Omega_D := \widetilde{\Omega} \setminus \Omega$. It typically plays the role of a nonlocal Dirichlet boundary in suitable settings. The case $\widetilde{\Omega} = \Omega$ implies $\Omega_D = \emptyset$ and thus the absence of further constraints. In this case, the resulting nonlocal problem can be interpreted as Neumann-type problem; see, e.g., [22, 31]. In Fig. 1, we present an exemplary configuration. Let $\delta > 0$ be an *interaction horizon*. We make the following assumption about the kernel function.

Assumption 1 We assume that for the matrix-valued *kernel function* $\mathbf{\Psi}_\delta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{n\times n}$, there exists an $s \in (0, 1)$ such that

$$\mathbf{\Psi}_\delta(\mathbf{x},\mathbf{y})\, \mathbb{1}_{B_\delta(\mathbf{x})}(\mathbf{y})\, |\mathbf{x}-\mathbf{y}|^{d+2s} \tag{1}$$

is bounded.

This allows singularities at the origin and also includes smooth kernels such as the constant kernel. Next, we introduce the *interaction neighborhood* $B_\delta(\mathbf{x})$, where

$$B_\delta(\mathbf{x}) := \{\mathbf{y} \in \mathbb{R}^d \mid |\mathbf{x}-\mathbf{y}|_\bullet \leqslant \delta\}, \tag{2}$$

for some norm $|\cdot|_\bullet$ in $\mathbb{R}^d$. We denote the truncated *kernel* by

$$\mathbf{C}_\delta(\mathbf{x},\mathbf{y}) := \mathbf{\Psi}_\delta(\mathbf{x},\mathbf{y})\mathbb{1}_{B_\delta(\mathbf{x})}(\mathbf{y}). \tag{3}$$

The linear *nonlocal operator* under consideration acting on a function $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^n$ is then given by

[2] There exists an open set $\mathcal{O} \subset \mathbb{R}^d$, such that $\mathcal{O} \cap \widetilde{\Omega} = \Omega$.

$$-\mathcal{L}_\delta \mathbf{u}(\mathbf{x}) := 2\int_{\widetilde{\Omega}} \big(\mathbf{C}_\delta(\mathbf{x},\mathbf{y})\mathbf{u}(\mathbf{x}) - \mathbf{C}_\delta(\mathbf{y},\mathbf{x})\mathbf{u}(\mathbf{y})\big)d\mathbf{y}. \tag{4}$$

Note that in general, the above integral does not exist, and we tacitly interpret the strong form of the operator in the Cauchy principal value sense[3] if necessary. By testing (4) with $\mathbf{v} : \widetilde{\Omega} \to \mathbb{R}^n$ where $\mathbf{v} = 0$ on Ω_D, we obtain the bilinear form

$$\begin{aligned} A(\mathbf{u},\mathbf{v}) &:= -\int_{\Omega} \mathbf{v}(\mathbf{x})^\top \mathcal{L}_\delta \mathbf{u}(\mathbf{x})d\mathbf{x} \\ &= 2\int_{\Omega} \mathbf{v}(\mathbf{x})^\top \int_{\widetilde{\Omega}} \mathbf{C}_\delta(\mathbf{x},\mathbf{y})\mathbf{u}(\mathbf{x}) \\ &\quad - \mathbf{C}_\delta(\mathbf{y},\mathbf{x})\mathbf{u}(\mathbf{y})d\mathbf{y}d\mathbf{x}. \end{aligned} \tag{5}$$

With $\mathbf{v} = \mathbf{0}$ on Ω_D and Fubini's theorem (see, e.g., [22]), the bilinear form can be written as

$$\begin{aligned} &A(\mathbf{u},\mathbf{v}) \\ &= \int_{\widetilde{\Omega}}\int_{\widetilde{\Omega}} (\mathbf{v}(\mathbf{x}) - \mathbf{v}(\mathbf{y}))^\top \big(\mathbf{C}_\delta(\mathbf{x},\mathbf{y})\mathbf{u}(\mathbf{x}) - \mathbf{C}_\delta(\mathbf{y},\mathbf{x})\mathbf{u}(\mathbf{y})\big)d\mathbf{y}d\mathbf{x} \\ &= \int_{\widetilde{\Omega}}\int_{\widetilde{\Omega}} \mathbb{1}_{B_\delta(\mathbf{x})}(\mathbf{y})(\mathbf{v}(\mathbf{x}) - \mathbf{v}(\mathbf{y}))^\top \big(\boldsymbol{\Psi}_\delta(\mathbf{x},\mathbf{y})\mathbf{u}(\mathbf{x}) - \boldsymbol{\Psi}_\delta(\mathbf{y},\mathbf{x})\mathbf{u}(\mathbf{y})\big)d\mathbf{y}d\mathbf{x}. \end{aligned} \tag{6}$$

Note that we exploited the symmetry of the exact indicator function $\mathbb{1}_{B_\delta(\mathbf{x})}(\mathbf{y})$ to obtain this equality.

Remark 2.1 (Nonlocal convection-diffusion). The kernel introduced in (3) is not assumed to be symmetric and may therefore exhibit a nonzero anti-symmetric component. A splitting of this kernel into its symmetric and anti-symmetric parts results in an additive splitting of the nonlocal operator defined in (4), say $\mathcal{L}_\delta = \mathcal{L}_\delta^d + \mathcal{L}_\delta^c$. Invoking the operators and terminology from the nonlocal vector calculus introduced in [8, 32], one can relate $-\mathcal{L}_\delta^d$ and $-\mathcal{L}_\delta^c$ to nonlocal diffusion and nonlocal convection, respectively (see, e.g., [26, Section 2.3]).

2.2 Finite Dimensional Approximation

For integers $K, M \in \mathbb{N}$, let $\mathcal{T}^h := \{\mathcal{E}_k\}_{k=1}^K$ denote a subdivision of $\widetilde{\Omega} = \Omega \cup \Omega_D$ into polyhedral finite elements with nodes $\{\mathbf{x}_m\}_{m=1}^M$.

Assumption 2 We assume that Ω and Ω_D can be exactly covered by the subdivisions $\mathcal{T}^h_\Omega = \{\mathcal{E}_k\}_{k=1}^{K_\Omega}$ and $\mathcal{T}^h_D = \{\mathcal{E}_k\}_{k=K_\Omega+1}^K$ with $\mathcal{T}^h = \mathcal{T}^h_\Omega \cup \mathcal{T}^h_D$, respectively, where $\mathcal{T}^h_D$ is possibly empty. Since we assume polyhedral elements, this implies that

$$\overline{\Omega} = \bigcup_{k=1}^{K_\Omega} \overline{\mathcal{E}}_k \quad \textit{and} \quad \overline{\Omega}_D = \bigcup_{k=K_\Omega+1}^{K} \overline{\mathcal{E}}_k \tag{7}$$

are polyhedral domains.

[3] More precisely, $\int_{\widetilde{\Omega}} h(\mathbf{x},\mathbf{y})d\mathbf{y} := \lim_{\epsilon\to 0^+} \int_{\widetilde{\Omega}\setminus B_\epsilon(\mathbf{x})} h(\mathbf{x},\mathbf{y})d\mathbf{y}$.

In the case of $\Omega_D = \emptyset$, we have that $K_\Omega = K$ and $\mathcal{T}^h = \mathcal{T}^h_\Omega$. For convenience of notation, we assume an ordering of the nodes such that $\{\mathbf{x}_m\}_{m=1}^{M_\Omega} \subset \Omega$ and $\{\mathbf{x}_m\}_{m=M_\Omega+1}^{M} \subset \overline{\Omega}_D$. This assumption is not made in the implementation of nlfem.

We implement scalar and vector-valued piecewise-linear continuous and discontinuous basis functions $\{\boldsymbol{\phi}_j\}_{j=1}^J$, where $J = M$ in the case of continuous and $J = (d+1)K$ in case of discontinuous basis functions. Again, for convenience, we assume an ordering of the basis functions and define the corresponding finite-dimensional subspaces

$$V^h(\widetilde{\Omega}, \mathbb{R}^n) := \operatorname{span}(\{\boldsymbol{\phi}_j\}_{j=1}^J), \text{ and } V_c^h(\widetilde{\Omega}, \mathbb{R}^n) := \operatorname{span}(\{\boldsymbol{\phi}_j\}_{j=1}^{J_\Omega}), \tag{8}$$

where $J_\Omega \leqslant J$ denotes the number of basis functions which have support within Ω. In case of continuous basis functions, the unknown coefficients correspond to the nodes lying in the interior of Ω with respect to $\widetilde{\Omega}$. See Fig. 1 for an illustration. Now, the evaluation of the bilinear form A on $V^h(\widetilde{\Omega}, \mathbb{R}^n) \times V_c^h(\widetilde{\Omega}, \mathbb{R}^n)$ can be written as sum over the finite elements, i.e.,

$$\begin{aligned} &A(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) = \\ &\sum_{k=1}^K \sum_{\ell=1}^K \left[\int_{\mathcal{E}_k} \int_{\mathcal{E}_\ell} (\boldsymbol{\phi}_i(\mathbf{x}) - \boldsymbol{\phi}_i(\mathbf{y}))^\top \big(\mathbf{C}_\delta(\mathbf{x},\mathbf{y}) \boldsymbol{\phi}_j(\mathbf{x}) - \mathbf{C}_\delta(\mathbf{y},\mathbf{x}) \boldsymbol{\phi}_j(\mathbf{y}) \big) d\mathbf{y} d\mathbf{x} \right]. \end{aligned} \tag{9}$$

Since the kernel $\mathbf{C}_\delta$ may exhibit a truncation on some pairs $(\mathcal{E}_k, \mathcal{E}_\ell)$, we need an appropriate approximation of its support. The number of elements in the interaction neighborhood of a point in 2d for a fixed horizon δ is in $\mathcal{O}(h^{-2})$, and a greater mesh size h can increase the sparsity in the discrete system. Therefore, a precise approximation can leverage efficiency, and it is desirable that the geometric error in the evaluation of (9) does not deteriorate the interpolation error of the finite element space. While the approximation of the infinity or ℓ^1 norm balls does not introduce a geometric error, the one for curved neighborhoods, like the Euclidean norm ball, does.

In the following, we describe two major examples of the implemented ball approximations for the Euclidean ball

$$B_\delta^2(\mathbf{x}) := \{\mathbf{y} \in \mathbb{R}^d \mid |\mathbf{x} - \mathbf{y}|_2 \leqslant \delta\},$$

which we call the *approxcaps* and the *nocaps* approximations. Both are based on the given finite element mesh $\mathcal{T}^h$. By “cap,” we mean the circular segments that arise when a finite element triangle is only partially covered by the Euclidean ball; see Fig. 2. Among others, these ball approximations are investigated in [1].

Definition 2.2 (*nocaps* Ball). For $\mathbf{x} \in \widetilde{\Omega}$, the *nocaps* ball approximation is defined as the convex hull of the intersection of the boundary $\partial B_\delta^2(\mathbf{x})$ of the Euclidean ball and the boundaries of the elements, i.e.,

$$B_\delta^{ncp}(\mathbf{x}) := \operatorname{conv}\left(\bigcup_{\mathcal{E}_\ell \in \mathcal{T}^h} \partial \mathcal{E}_\ell \cap \partial B_\delta^2(\mathbf{x}) \right),$$

where the dependency on the mesh size h is omitted in the notation.

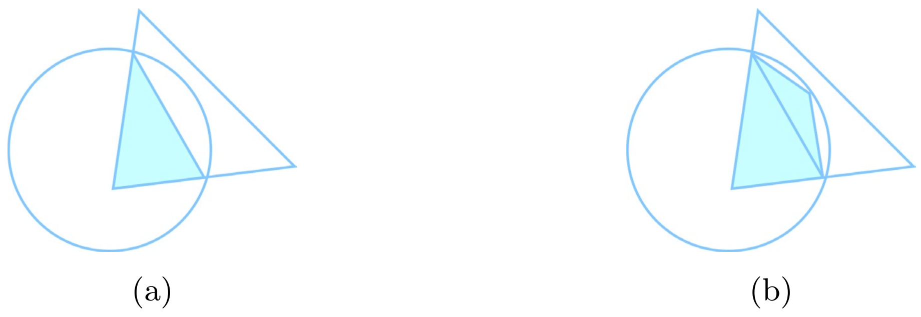
Fig. 2 If a finite element triangle is only partially covered by an Euclidean ball, the intersection contains circular caps. The *nocaps* ball (**a**) omits this cap, whereas the *approxcaps* ball (**b**) retriangulates the whole intersection

For $d = 2$, one can show that the area of the symmetric difference of $B_\delta^2(\mathbf{x})$ and $B_\delta^{ncp}(\mathbf{x})$ is provably of order $\mathcal{O}(h^2)$ when the mesh size h tends to zero. This is error commensurate with respect to the interpolation error of a linear finite element ansatz space; details for the latter two statements can be found in [1]. The convex hull omits circular caps which appear in the intersection of some elements with the neighborhood $\mathcal{E}_\ell \cap B_\delta^2(\mathbf{x})$ and have a significant size on coarse grids. Therefore, by adding additional points on the center of possible caps, the geometric error can be reduced even further while still being in $\mathcal{O}(h^2)$ for $d = 2$; see again Fig. 2. The maximum number of caps for a single intersection $\mathcal{E}_\ell \cap B_\delta^2(\mathbf{x})$ is three.

While the results in [1] are derived for a fixed horizon δ, related investigations for the local limit ($\delta \to 0$) with polygonal ball approximations can be found in [33].

Definition 2.3 (*approxcaps* Ball). Let $\mathbf{x} \in \widetilde{\Omega}$. We denote the points on the cap center of each nonempty intersection $\mathcal{E}_\ell \cap \partial B_\delta^2(\mathbf{x})$ by $\mathbf{y}_\ell$. The *approxcaps* ball is then defined by

$$B_\delta^{acp}(\mathbf{x}) := \\ \operatorname{conv}\left(B_\delta^{ncp}(\mathbf{x}) \cup \{\mathbf{y}_\ell \mid \mathbf{y}_\ell \textit{ cap center of } \mathcal{E}_\ell \cap \partial B_\delta^2(\mathbf{x}) \textit{ for some } \mathcal{E}_\ell \in \mathcal{T}^h\}\right).$$

Exact quadrature rules for circular caps can be found in [34]. Here, however, the quadrature points have to be computed during run time, as the rules depend on the geometry of the cap for higher quadrature orders. Therefore, we do not consider these exact rules in nlfem.

In addition to these approximate balls, we next also introduce the infinity norm ball.

Definition 2.4 (Infinity Norm Ball). For $\mathbf{x} \in \widetilde{\Omega}$, the infinity normball is defined by

$$B_\delta^\infty(\mathbf{x}) := \{\mathbf{y} \in \mathbb{R}^d \mid |\mathbf{x} - \mathbf{y}|_\infty \leqslant \delta\}.$$

Proofs for the convergence of the nonlocal Dirichlet-type problem to the classical Dirichlet problem with corresponding scaling constants for various kernel functions can be found, e.g., in [26]. Since the infinity normball is implemented exactly, it allows numerical tests of the expected asymptotic compatibility of the discretization scheme [30].

We finally note that the indicator function based on any implemented truncation may lack symmetry. More precisely, there might exist $\mathbf{x}$ and $\mathbf{y}$, for which

$$\mathbb{1}_{B_\delta^\#(\mathbf{x})}(\mathbf{y}) \neq \mathbb{1}_{B_\delta^\#(\mathbf{y})}(\mathbf{x}),$$

where $B_\delta^\#(\mathbf{x})$, $\# \in \{ncp, acp, \infty\}$, represents one of the implemented truncations. This artifact stems from the ball approximation itself in the case of the *nocaps* and *approxcaps* ball, but can also be caused by the quadrature; see Remark 2.6 below.

We define the integrand

$$\Phi_{ij}(\mathbf{x},\mathbf{y}) := (\boldsymbol{\phi}_i(\mathbf{x}) - \boldsymbol{\phi}_i(\mathbf{y}))^\top(\boldsymbol{\Psi}_\delta(\mathbf{x},\mathbf{y})\boldsymbol{\phi}_j(\mathbf{x}) - \boldsymbol{\Psi}_\delta(\mathbf{y},\mathbf{x})\boldsymbol{\phi}_j(\mathbf{y})) \tag{10}$$

and, based on the ball approximation and (6), the *approximate bilinear form*

$$\begin{aligned} A_h^\#(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) &:= \int_{\widetilde{\Omega}} \int_{\widetilde{\Omega}} \mathbb{1}_{B_\delta^\#(\mathbf{x})}(\mathbf{y}) \Phi_{ij}(\mathbf{x},\mathbf{y}) d\mathbf{y} d\mathbf{x} \\ &= \sum_{k=1}^{K} \sum_{\ell=1}^{K} \int_{\mathcal{E}_k} \int_{\mathcal{E}_\ell} \mathbb{1}_{B_\delta^\#}(\mathbf{x},\mathbf{y}) \Phi_{ij}(\mathbf{x},\mathbf{y}) d\mathbf{y} d\mathbf{x}. \end{aligned} \tag{11}$$

By Fubini's integration theorem, for sufficiently smooth basis functions $\boldsymbol{\phi}_j \in V^h(\widetilde{\Omega}, \mathbb{R}^n)$ and $\boldsymbol{\phi}_i \in V_c^h(\widetilde{\Omega}, \mathbb{R}^n)$, the approximate bilinear form defined in (11) can be written as

$$\begin{aligned} & A_h^\#(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) \\ &= \int_{\widetilde{\Omega}} \int_{\widetilde{\Omega}} \mathbb{1}_{B_\delta^\#(\mathbf{x})}(\mathbf{y}) \Phi_{ij}(\mathbf{x},\mathbf{y}) d\mathbf{y} d\mathbf{x} \\ &= \int_{\Omega} \int_{\widetilde{\Omega}} \mathbb{1}_{B_\delta^\#(\mathbf{x})}(\mathbf{y}) \boldsymbol{\phi}_i(\mathbf{x})^\top \big(\boldsymbol{\Psi}_\delta(\mathbf{x},\mathbf{y})\boldsymbol{\phi}_j(\mathbf{x}) - \boldsymbol{\Psi}_\delta(\mathbf{y},\mathbf{x})\boldsymbol{\phi}_j(\mathbf{y})\big) d\mathbf{y}\, d\mathbf{x} \\ &\quad - \int_{\widetilde{\Omega}} \int_{\Omega} \mathbb{1}_{B_\delta^\#(\mathbf{x})}(\mathbf{y}) \boldsymbol{\phi}_i(\mathbf{y})^\top \big(\boldsymbol{\Psi}_\delta(\mathbf{x},\mathbf{y})\boldsymbol{\phi}_j(\mathbf{x}) - \boldsymbol{\Psi}_\delta(\mathbf{y},\mathbf{x})\boldsymbol{\phi}_j(\mathbf{y})\big) d\mathbf{y}\, d\mathbf{x} \\ &= 2 \int_{\Omega} \boldsymbol{\phi}_i(\mathbf{x})^\top \int_{\widetilde{\Omega}} \mathbb{1}^S_{B_\delta^\#}(\mathbf{x},\mathbf{y}) \big(\boldsymbol{\Psi}_\delta(\mathbf{x},\mathbf{y})\boldsymbol{\phi}_j(\mathbf{x}) - \boldsymbol{\Psi}_\delta(\mathbf{y},\mathbf{x})\boldsymbol{\phi}_j(\mathbf{y})\big) d\mathbf{y}\, d\mathbf{x}, \end{aligned}$$

where

$$\mathbb{1}^S_{B_\delta^\#}(\mathbf{x},\mathbf{y}) := \frac{1}{2}\Big(\mathbb{1}_{B_\delta^\#(\mathbf{x})}(\mathbf{y}) + \mathbb{1}_{B_\delta^\#(\mathbf{y})}(\mathbf{x})\Big). \tag{12}$$

In view of (5), this shows that the approximate bilinear form $A_h^\#$ can be interpreted as the discretization of the operator $-\mathcal{L}_\delta$ based on the symmetrified approximate indicator function $\mathbb{1}^S_{B_\delta^\#}(\mathbf{x},\mathbf{y})$ instead of $\mathbb{1}_{B_{\delta(\mathbf{x})}}(\mathbf{y})$ in the strong form. However, defining the approximate bilinear form as in (11) guarantees the symmetry of the stiffness matrix $(A_h^\#(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i))_{i,j}$; see also Remark 2.6 below.

In the following, we discuss the evaluation of the local contributions

$$A_{k\ell}^\#(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) := \int_{\mathcal{E}_k} \int_{\mathcal{E}_\ell} \mathbb{1}_{B_\delta^\#(\mathbf{x})}(\mathbf{y}) \Phi_{ij}(\mathbf{x},\mathbf{y}) d\mathbf{y}\, d\mathbf{x} \tag{13}$$

to the (i, j)-th entry of the stiffness matrix.

2.3 Population of the Stiffness Matrix

The assembly algorithm iterates over all pairs of elements and then adds the local contributions (13) to the stiffness matrix. We call a pair of elements $(\mathcal{E}_k, \mathcal{E}_\ell)$ *intersecting* if

$$\overline{\mathcal{E}_k} \cap \overline{\mathcal{E}_\ell} \neq \emptyset.$$

If a pair is not intersecting, we call it *disjoint*. For the evaluation of the contributions, it is only important how the kernel behaves on a fixed pair of elements. If for example the kernel exhibits a singularity at the origin and the pair is disjoint, the singularity does not occur. Also, if a pair of elements $(\mathcal{E}_k, \mathcal{E}_\ell)$ fulfills that

$$\mathcal{E}_\ell \subset B_\delta(\mathbf{x}) \text{ for all } \mathbf{x} \in \mathcal{E}_k,$$

the truncation does not come into play. We therefore distinguish two cases. In the first case, the kernel has a singularity with $s < 0.5$ or the pair of elements is disjoint. This allows to derive the representation (14) of the bilinear form because the singularity either is not too strong or does not occur at all. In the opposite case, the kernel has a singularity with $s \geqslant 0.5$ and the elements are intersecting so that the expression (13) cannot be divided into smaller parts.

2.3.1 Disjoint Pairs or Kernels with $s < 0.5$

If $s < 0.5$ in (1) or if the pair $(\mathcal{E}_k, \mathcal{E}_\ell)$ is disjoint, the local contributions (13) can be factored out and computed separately, i.e.,

$$\begin{aligned}
A^{\#}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) = &\int_{\mathcal{E}_k}\int_{\mathcal{E}_\ell} \mathbb{1}_{B^{\#}_\delta(\mathbf{x})}(\mathbf{y})\boldsymbol{\phi}_i(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Psi}_\delta(\mathbf{x},\mathbf{y})\boldsymbol{\phi}_j(\mathbf{x})d\mathbf{y}\, d\mathbf{x} \\
&- \int_{\mathcal{E}_k}\int_{\mathcal{E}_\ell} \mathbb{1}_{B^{\#}_\delta(\mathbf{x})}(\mathbf{y})\boldsymbol{\phi}_i(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Psi}_\delta(\mathbf{y},\mathbf{x})\boldsymbol{\phi}_j(\mathbf{y})d\mathbf{y}\, d\mathbf{x} \\
&+ \int_{\mathcal{E}_k}\int_{\mathcal{E}_\ell} \mathbb{1}_{B^{\#}_\delta(\mathbf{x})}(\mathbf{y})\boldsymbol{\phi}_i(\mathbf{y})^{\mathsf{T}}\boldsymbol{\Psi}_\delta(\mathbf{y},\mathbf{x})\boldsymbol{\phi}_j(\mathbf{y})d\mathbf{y}\, d\mathbf{x} \\
&- \int_{\mathcal{E}_k}\int_{\mathcal{E}_\ell} \mathbb{1}_{B^{\#}_\delta(\mathbf{x})}(\mathbf{y})\boldsymbol{\phi}_i(\mathbf{y})^{\mathsf{T}}\boldsymbol{\Psi}_\delta(\mathbf{x},\mathbf{y})\boldsymbol{\phi}_j(\mathbf{x})d\mathbf{y}\, d\mathbf{x}. \\
=: &A^{\#_1}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) + A^{\#_2}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) \\
&+ A^{\#_3}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) + A^{\#_4}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i).
\end{aligned} \tag{14}$$

In (14), the basis functions, and not their differences, are integrated. Consequently, respective terms in (14) yield identical values for linear discontinuous and continuous elements, and it becomes apparent that, if $s < 0.5$, the bilinear form can also be evaluated on discontinuous ansatz spaces. Therefore, nlfem allows continuous and discontinuous finite element spaces if $s < 0.5$. In fact, the splitting (14) is not viable for $s \geqslant 0.5$ where we require some regularity[4] in $\boldsymbol{\phi}_i$ and $\boldsymbol{\phi}_j$ for the integral to exist.

The expression $A^{\#_1}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i)$ is nonzero only if the element $\mathcal{E}_k$ lies in the support of $\boldsymbol{\phi}_i$ and $\boldsymbol{\phi}_j$. Similarly, the contribution of $A^{\#_3}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i)$ is linked to the element $\mathcal{E}_\ell$. The term

[4] For example, Lipschitz continuity of the basis functions $\boldsymbol{\phi}_i$ and $\boldsymbol{\phi}_j$ suffices.

Algorithm 1 Evaluate $A^{\#}_{k\ell}$ in (13)

```
1: if $\mathcal{E}_k, \mathcal{E}_\ell$ intersect and the kernel is singular then
2:     Evaluate $A^{\#}_{k\ell}$ via quadrature for singular kernels;
3:     see Section 2.4.2 below
4: else
5:     Evaluate $A^{\#}_{k\ell}$ via quadrature for kernel truncations;
6:     see Section 2.4.1 below
7: end if
```

$A^{\#_2}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i)$ is nonzero only if $\boldsymbol{\phi}_i$ has its support on $\mathcal{E}_k$ and $\boldsymbol{\phi}_j$ on $\mathcal{E}_\ell$, where the converse holds for $A^{\#_4}_{k\ell}(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i)$. That way, we derive the indices of the basis functions corresponding to the pair $(\mathcal{E}_k, \mathcal{E}_\ell)$ in the stiffness matrix. We note that the same indices of basis functions occur again for the pair $(\mathcal{E}_\ell, \mathcal{E}_k)$, but the contributions to the stiffness matrix are not identical as the truncation $\mathbb{1}_{B^{\#}_\delta(\mathbf{x})}(\mathbf{y})$ is not symmetric. Ultimately, the contribution of the two pairs $(\mathcal{E}_k, \mathcal{E}_\ell)$ and $(\mathcal{E}_\ell, \mathcal{E}_k)$ together lead to the symmetrified truncation (12).

2.3.2 Kernels with $s \geqslant 0.5$ on Intersecting Pairs

If a pair of elements is intersecting and $s \geqslant 0.5$, the separation of integrands in (14) is not admissible. Therefore, nlfem is restricted to continuous basis functions if $s \geqslant 0.5$. As the pair is intersecting, the elements are either vertex touching, edge touching, or identical. Therefore, the number of basis functions to be considered in these cases is 5, 4, or 3, respectively. If we denote the vertices by $j_1, \dots, j_\ell$ for $\ell = 5, 4, 3$, we obtain 25, 16, or 9 pairs of basis functions $(\boldsymbol{\phi}_{j_\nu}, \boldsymbol{\phi}_{j_{\nu'}})$ for $\nu, \nu' = 1, \dots, \ell$ which yield nonzero contributions to the local stiffness matrix $(A^{\#}_{k\ell}(\boldsymbol{\phi}_{j_\nu}, \boldsymbol{\phi}_{j_{\nu'}}))_{1 \leqslant \nu, \nu' \leqslant \ell}$.

2.4 Quadrature

The quadrature rules need to work for kernels with truncations and singularities. However, the implemented quadrature rules in nlfem do not account for both at the same time on a fixed pair of elements. We therefore require the following assumption.

Assumption 3 The quadrature rules for singular kernels assume that for all intersecting pairs of elements $(\mathcal{E}_k, \mathcal{E}_\ell)$, it holds that $\mathcal{E}_\ell \subset B_\delta(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{E}_k$.

If Assumption 3 is violated, the interaction neighborhood of the kernel is overestimated. This deteriorates the error finite element error. However, for sufficiently small mesh size, say $3h \leqslant \delta$, the problem does not occur.

Remark 2.5 Specific quadrature rules for singular kernels are required for sufficiently strong singularities only. Kernels like the peridynamics kernel (28) can be integrated by simply technically avoiding zero-divisions. More precisely, by choosing fixed quadrature rules for the inner and outer integral which do not have shared quadrature points, we make sure that $(\mathbf{x} - \mathbf{y}) \neq \mathbf{0}$ in (28) for any kernel evaluation during the assembly process.

Given Assumption 3 and the fact that disjoint pairs do not require a treatment with regularizing integral transforms [16], we can evaluate all contributions as given in Algorithm 1.

In any case, the quadrature is performed by pulling back the domain of integration to a reference domain

$$\widehat{\mathcal{E}} \times \widehat{\mathcal{E}}, \quad \text{where} \quad \widehat{\mathcal{E}} := \{\widehat{\mathbf{x}} \in \mathbb{R}^d \mid \widehat{\mathbf{x}} \geqslant 0, \sum_{\iota=1}^{d} \widehat{\mathbf{x}}_\iota \leqslant 1\}.$$

The affine linear mapping $\chi_k : \widehat{\mathcal{E}} \to \mathcal{E}$ with Jacobian determinant $|\chi_k|$ from the reference to a physical element allows to define the pullback

$$\widehat{\Phi}_{k\ell,ij}(\widehat{\mathbf{x}}, \widehat{\mathbf{y}}) := \Phi_{ij}(\chi_k(\widehat{\mathbf{x}}), \chi_\ell(\widehat{\mathbf{y}}))|\chi_k||\chi_\ell|,$$

the inner integral

$$\widehat{\mathcal{K}}^{\#}_{k\ell,ij}(\widehat{\mathbf{x}}) := \int_{\widehat{\mathcal{E}}} \mathbb{1}_{B^{\#}_\delta}(\chi_k(\widehat{\mathbf{x}}), \chi_\ell(\widehat{\mathbf{y}}))\widehat{\Phi}_{k\ell,ij}(\widehat{\mathbf{x}}, \widehat{\mathbf{y}})d\widehat{\mathbf{y}} \tag{15}$$

and the local contribution to the (i, j)-th entry of the stiffness matrix

$$\widehat{A}^{\#}_{k\ell,ij} := \int_{\widehat{\mathcal{E}}} \widehat{\mathcal{K}}^{\#}_{k\ell,ij}(\widehat{\mathbf{x}})d\widehat{\mathbf{x}} \tag{16}$$

on the reference element.

2.4.1 Quadrature for Kernel Truncations

For some pairs $(\mathcal{E}_k, \mathcal{E}_\ell)$, we find that $\mathcal{E}_\ell$ is only partially covered by the interaction neighborhood $B^{\#}_\delta(\mathbf{x})$ for some $\mathbf{x} \in \mathcal{E}_k$, so that $\mathcal{E}_\ell \cap B^{\#}_\delta(\mathbf{x}) \subsetneq \mathcal{E}_\ell$. In this case, the ball approximations $B^{ncp}_\delta(\mathbf{x})$ and $B^{acp}_\delta(\mathbf{x})$ as well as the ball $B^{\infty}_\delta(\mathbf{x})$ require a retriangulation of the integration domain $\mathcal{E}_\ell \cap B^{\#}_\delta(\mathbf{x})$. We denote the set of elements which result from such a retriangulation[5] by

$$\mathcal{T}^{\#}_{h,\ell}(\mathbf{x}) := \{\tilde{\mathcal{E}}_{\tilde{\ell}}\}_{\tilde{\ell}=1}^{L_\ell},$$

so that $\bigcup \mathcal{T}^{\#}_{h,\ell}(\mathbf{x}) = \mathcal{E}_\ell \cap B^{\#}_\delta(\mathbf{x})$. In view of Fig. 2, the set $\mathcal{T}^{\#}_{h,\ell}(\mathbf{x})$ collects the elements making up the shaded region $\bigcup \mathcal{T}^{\#}_{h,\ell}(\mathbf{x})$ for the *nocaps* ball (Fig. 2a) and the *approxcaps* ball (Fig. 2b), respectively. Let $\{\widehat{\mathbf{y}}_q, d_{\widehat{\mathbf{y}}_q}\}_{q=1}^{Q}$ denote a quadrature rule on the reference element $\widehat{\mathcal{E}}$, and let $\widehat{\mathbf{x}} \in \widehat{\mathcal{E}}$ be some reference point. Then, the fully discrete inner integral from (15) reads as

$$\widehat{\mathcal{K}}^{\#}_{k\ell,ij}(\widehat{\mathbf{x}}) \approx \sum_{\tilde{\ell}=1}^{L_\ell} \sum_{q=1}^{Q} \widehat{\Phi}_{k\tilde{\ell},ij}(\widehat{\mathbf{x}}, \widehat{\mathbf{y}}_q)d_{\widehat{\mathbf{y}}_q}. \tag{17}$$

In finite element implementations, the function values of basis functions are usually precomputed at the quadrature points and stored. However, if a retriangulation is necessary, the physical coordinates of the quadrature points $\chi_{\tilde{\ell}}(\widehat{\mathbf{y}}_q)$ for $q = 1, \dots, Q$ on some element $\tilde{\mathcal{E}}_{\tilde{\ell}} \in \mathcal{T}^{\#}_{h,\ell}(\mathbf{x})$ are known at runtime only and the basis functions are evaluated at the corresponding points.

[5] Note that these elements can be strict subsets of finite elements and should not be confused with the original finite element mesh. This is why additional notation is needed.

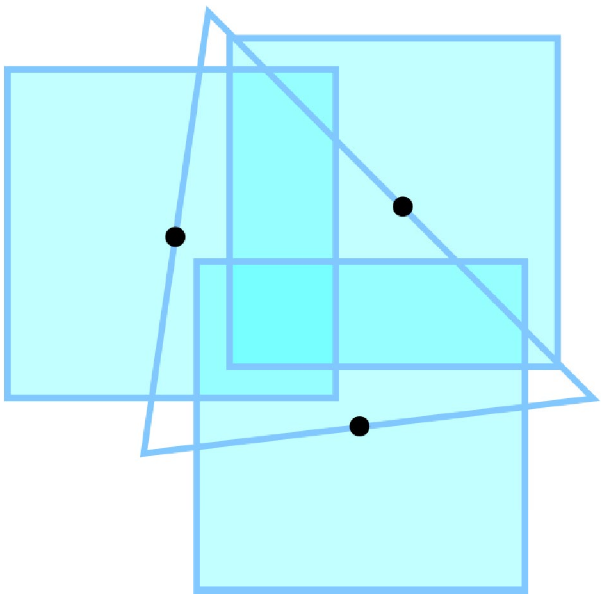
Fig. 3 The figure shows the approximation of the interaction domain of a single triangular element by interaction neighborhoods of three quadrature points for the infinity normball B_δ^∞

Now, let $\{\widehat{\mathbf{x}}_p, d_{\widehat{\mathbf{x}}_p}\}_{p=1}^P$ be a quadrature rule with points $\widehat{\mathbf{x}}_p$ and weights $d_{\widehat{\mathbf{x}}_p}$ for the reference element of the outer integral. Then, with (17), the discretized version of the local contribution to the (i,j)–th entry of the stiffness matrix (16) is obtained by

$$\widehat{A}^{\#}_{k\ell,ij} \approx \sum_{p=1}^{P} \left(\sum_{\tilde{\ell}=1}^{L_\ell} \sum_{q=1}^{Q} \widehat{\Phi}_{k\tilde{\ell},ij}(\widehat{\mathbf{x}}_p, \widehat{\mathbf{y}}_q) d_{\widehat{\mathbf{y}}_q} \right) d_{\widehat{\mathbf{x}}_p}.$$

2.4.2 Quadrature for Singularities

Some kernel functions exhibit a singularity at the origin, which lies in the integration domain of $\widehat{\mathcal{K}}^{\#}_{k\ell,ij}(\widehat{\mathbf{x}})$ whenever the pair $(\mathcal{E}_k, \mathcal{E}_\ell)$ intersects. We therefore require regularizing integral transforms [16]. Assumption 3 allows to ignore possible truncations in (16) and to simply evaluate

$$\widehat{A}^{\#}_{k\ell,ij} = \int_{\widehat{\mathcal{E}}} \int_{\widehat{\mathcal{E}}} \widehat{\Phi}_{k\ell,ij}(\widehat{\mathbf{x}}, \widehat{\mathbf{y}}) d\widehat{\mathbf{y}} d\widehat{\mathbf{x}}. \tag{18}$$

Intersecting pairs $(\mathcal{E}_\ell, \mathcal{E}_k)$ can be vertex touching, edge touching, or identical. For each of those cases, we apply integral transforms, which again pull back subsets of the integration domain $\widehat{\mathcal{E}} \times \widehat{\mathcal{E}}$ to the unit cube $(0,1)^4$. The transformations are well established and applied, for example, in the field of boundary element methods. Details can be found in [16, 17]. Note again, that in the case of singular kernels with $s \geqslant 0.5$, the implemented routines cannot evaluate the expression in (18) for discontinuous basis functions.

Remark 2.6 (Asymmetry due to quadrature). We have mentioned in Sect. 2.2 that the approximate indicator function based on the *approxcaps* and *nocaps* balls may generally lack in symmetry and therefore would lead to nonsymmetric stiffness matrices if one used representation (5) of the bilinear form. Also, truncations invoked by the infinity norm ball $B_\delta^\infty(\mathbf{x})$, which can be implemented exactly, can be nonsymmetric on sufficiently irregular grids. These two observations hold true *independent* of the symmetry of the kernel function $\boldsymbol{\Psi}_\delta(\mathbf{x}, \mathbf{y})$. However, it is a desirable feature that the symmetry of the kernel, i.e., the self-adjointness of the operator, is transported through the discretization process. In other words, for a symmetric kernel, we expect a symmetric stiffness matrix. This behavior is more intricate and related to the approximation of the interaction domain of a finite element by the union of interaction neighborhoods of quadrature points; see Fig. 3. Thus, we use by default representation (11) instead, which as stated above corresponds in strong form to the operator $-\mathcal{L}_\delta$ based on the approximate indicator function (12). Thereby, we guarantee the stiffness matrix to be symmetric up to machine precision for any symmetric kernel function. Also, an important consequence of the symmetrification is that the null space of the stiffness matrix related to pure Neumann-type problems contains the constant vectors up to machine precision. This allows the application of projected Krylov subspace methods or rank-1 corrections to efficiently evaluate a pseudoinverse of the stiffness matrix.

We summarize the different cases for the quadrature of kernels in Table 1. The table distinguishes between different degrees of singularity s in the case of $d = 2$ as well as intersecting and disjoint pairs of elements. When $s > 0$, we apply regularizing integral transforms represented by $\clubsuit$ in the table. This quadrature rule is applied for intersecting pairs and requires Assumption 3. Whenever $s < 0.5$, we allow for discontinuous and continuous basis functions. When $s \geqslant 0.5$, only continuous basis functions are provided. For more details, we refer to [18]. If $-1 < s \leqslant 0$, the singularity might be so weak that it suffices to simply avoid zero-divisions on intersecting pairs, which is represented by $\diamondsuit$ in the table. However, the quadrature rule $\clubsuit$ can also be applied. If $s \leqslant -1$, there is no singularity at all, and the symbol $\heartsuit$ represents a standard quadrature rule. The rules, $\heartsuit$ and $\diamondsuit$, respect the kernel truncation, i.e., they do not require Assumption 3.

2.5 Traversal of the Interaction Neighborhood

The local contributions $\widehat{A}^{\#}_{k\ell,ij}$ to the stiffness matrix, defined in (13), can be nonzero even for pairs of disjoint finite elements $(\mathcal{E}_k, \mathcal{E}_\ell)$, which we then refer to as *interacting* elements in the following. Unstructured meshes and the various supports of kernel functions call for a flexible routine to identify interacting elements. The identification can be accomplished if the assembly follows a breadth-first search. A breadth-first search consecutively traverses the nodes in a graph starting in a root node. It first visits the

Table 1 The table gives an overview of the different quadrature rules which can be applied for dimension $d = 2$ in nlfem

	$s \leqslant -1$	$-1 < s \leqslant 0$	$0 < s < 0.5$	$0.5 < s < 1$
Intersecting	$\heartsuit$	$\diamondsuit$ or $\clubsuit$	$\clubsuit$	$\clubsuit$
Disjoint	$\heartsuit$	$\heartsuit$	$\heartsuit$	$\heartsuit$
	CG and DG			CG

$\heartsuit$ Standard quadrature rule on $\widehat{\mathcal{E}} \times \widehat{\mathcal{E}}$. $\diamondsuit$ Quadrature rule on $\widehat{\mathcal{E}} \times \widehat{\mathcal{E}}$ that avoids $\widehat{\mathbf{x}}_p = \widehat{\mathbf{y}}_q$. $\clubsuit$ Quadrature rule on $(0, 1)^4$ after regularizing integral transforms nodes which are directly adjacent to the root node. Then, only it visits the (unvisited) neighbors of the successive nodes. The search stops when a certain criterion is met or no further successors can be found. We refer to the neighbors of the root node as *first layer* of the search.

In order to describe the algorithm, we define the set of immediate neighbors of some element $\mathcal{E}_k$ by

$$\mathcal{N}(\mathcal{E}_k) = \{\mathcal{E}_\ell \mid \overline{\mathcal{E}_k} \cap \overline{\mathcal{E}_\ell} \neq \emptyset\} \tag{19}$$

and the *adjacency graph* $\mathbb{T}_{adj} := \left(\mathcal{T}^h, E_{adj}\right)$ of the finite element mesh with vertices $\mathcal{T}^h$ and edges

$$E_{adj} := \{(\mathcal{E}_k, \mathcal{E}_\ell) \in \mathcal{T}^h \times \mathcal{T}^h \mid \mathcal{E}_k \in \mathcal{N}(\mathcal{E}_\ell)\}.$$

The graph $\mathbb{T}_{adj}$ can be understood as the dual graph of the finite element mesh, and its vertices are therefore given by the elements. We additionally define the *interaction graph* $\mathbb{T}_S := \left(\mathcal{T}^h, E_S\right)$ with vertices $\mathcal{T}^h$ and edges

$$E_S := \left\{ (\mathcal{E}_k, \mathcal{E}_\ell) \in \mathcal{T}^h \times \mathcal{T}^h \,\middle|\, \widehat{A}^{\#}_{k\ell,ij} \neq 0 \text{ for some } \boldsymbol{\phi}_j, \boldsymbol{\phi}_i \in V^h(\widetilde{\Omega}, \mathbb{R}^n) \right\}. \tag{20}$$

The set of edges E_{adj} is contained in E_S because touching elements interact. Hence, the set of vertices of $\mathbb{T}_{adj}$ and $\mathbb{T}_S$ are identical, and all edges of $\mathbb{T}_{adj}$ are contained in $\mathbb{T}_S$. In that sense, $\mathbb{T}_{adj}$ is a subgraph of $\mathbb{T}_S$. Furthermore, for a fixed element $\mathcal{E}_k$, let us denote the set of all interacting elements by $\mathcal{T}^h_k := \{\mathcal{E}_\ell \in \mathcal{T}^h \big| (\mathcal{E}_k, \mathcal{E}_\ell) \in E_S\}$. We then define the subgraph $\mathbb{T}_{S_k} := \left(\mathcal{T}^h_k, E_{S_k}\right)$ of $\mathbb{T}_S$ with vertices $\mathcal{T}^h_k$ and edges given by

$$E_{S_k} := \left\{ (\mathcal{E}_k, \mathcal{E}_\ell) \in \mathcal{T}^h \times \mathcal{T}^h \,\middle|\, (\mathcal{E}_k, \mathcal{E}_\ell) \in E_S \right\}.$$

The adjacency graph $\mathbb{T}_{adj}$ can be computed and stored efficiently, while the interaction graph $\mathbb{T}_S$ exhibits storage requirements which are comparable to the full stiffness matrix. We also note that the breadth-first search described below allows to naturally identify intersecting pairs $(\mathcal{E}_k, \mathcal{E}_\ell)$, as they only occur in the first layer of the traversal. This important built-in feature is used to identify the intersection cases for element pairs mentioned in Sect. 2.3.

Assumption 4 We assume without loss of generality that each of the graphs $\mathbb{T}_{S_k}$, $\mathbb{T}_{adj}$, and $\mathbb{T}_S$ is connected.

It is clear that Assumption 4 can be violated if $\widetilde{\Omega}$ is not connected or even if it is connected as depicted in Fig. 4a. As a remedy, the implementation allows to add artificial vertices and elements to the mesh which connect certain parts of the mesh. The artificial vertices and elements allow to change the mesh topology so that it is guaranteed that the interaction domain of an element, and hence $\mathbb{T}_{S_m}$, is always connected; see Fig. 4b. Of course, the artificial elements do not enter any integration routines. A straightforward option is to embed $\widetilde{\Omega}$ into a bounded and convex *hold-all domain* $\widehat{\Omega} \subset \mathbb{R}^n$, which guarantees that the graphs $\mathbb{T}_{S_k}$, $\mathbb{T}_{adj}$, and $\mathbb{T}_S$ are always connected; see Fig. 4b.

In that sense, Assumption 4 does not cause any loss of generality. If Assumption 4 holds, we can recover the subgraph $\mathbb{T}_{S_k}$ with a truncated breadth-first traversal of $\mathbb{T}_{adj}$ starting in the root node $\mathcal{E}_k$ as given in Algorithm 2. To that end, we define an empty queue Q

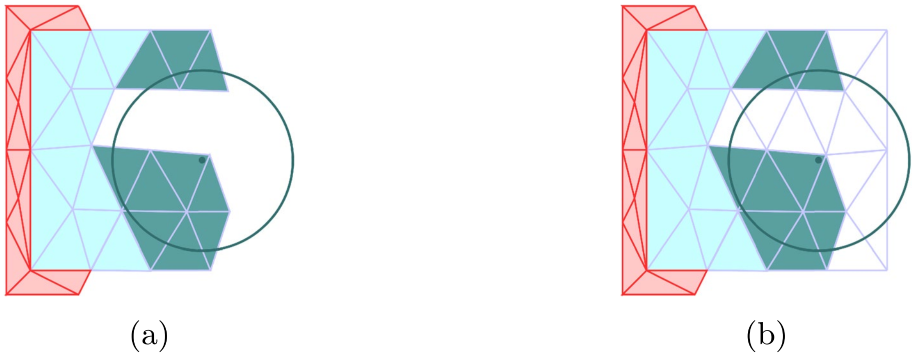

Fig. 4 Connected domain causing disconnected neighborhoods. In **a**, the interaction neighborhood of an element is disconnected even though the domain is connected. In **b**, a convex hold-all domain $\widehat{\Omega}$ allows to account for possible interactions within disconnected interaction neighborhoods

of elements to which we can append new elements and read and remove them in a first-in first-out ordering. As first step, the root node $\mathcal{E}_k$ is appended to $Q = [\mathcal{E}_k]$. While Q is not empty, an element $\mathcal{E}_{\widetilde{\ell}}$ is read and removed from the queue in a first-in first-out order. Its immediate neighbor, say $\mathcal{E}_\ell \in \mathcal{N}(\mathcal{E}_{\widetilde{\ell}})$, are obtained from the adjacency graph and the integrals $\widehat{A}^{\#}_{k\ell,ij}$ are successively evaluated for all $\mathcal{E}_\ell \in \mathcal{N}(\mathcal{E}_{\widetilde{\ell}})$. The element $\mathcal{E}_\ell$ is added to the queue whenever the integral does not vanish, and finally marked as visited. This procedure automatically truncates the search to interacting elements for any connected interaction neighborhood.

3 Numerical Examples

We solve a truncated fractional-type steady-state diffusion problem, a linear bond-based peridynamics equation [5, 35] and a steady-state diffusion problem based on the infinity normball. In the first two examples, we demonstrate the convergence rate of the approximate solutions to a manufactured solution as the mesh size $h \to 0$. In the latter example, we demonstrate the asymptotic compatibility of the discretization scheme for a vanishing horizon δ.

To this end, let $\Omega := (0, 0.5)^2 \subset \mathbb{R}^2$. We define the interaction domain of Ω by $\Omega_D := [-\delta,\ 0.5 + \delta]^2 \setminus \Omega$, so that $\widetilde{\Omega} = [-\delta, 0.5 + \delta]^2$. We then want to solve the nonlocal Dirichlet-type problem

Algorithm 2 Traversal of interaction neighborhood

```
1:  for $\mathcal{E}_k$ in $\mathcal{T}^h$ in parallel threads do
2:      Append $\mathcal{E}_k$ to the queue $Q$
3:      while $Q$ is not empty do
4:          Read and remove $\mathcal{E}_{\widetilde{\ell}}$ from $Q$
5:          for $\mathcal{E}_\ell$ in local neighborhood $\mathcal{N}(\mathcal{E}_{\widetilde{\ell}})$ do
6:              if $\mathcal{E}_\ell$ is not visited then
7:                  Evaluate $A^{\#}_{k\ell}$ (Algorithm 1)
8:                  if $A^{\#}_{k\ell} \neq 0$ append $\mathcal{E}_\ell$ to $Q$
9:                  Mark $\mathcal{E}_\ell$ as visited
10:             end if
11:         end for
12:     end while
13: end for
```

$$\begin{cases} -\mathcal{L}_\delta \mathbf{u} = \mathbf{f} & \text{in } \Omega, \\ \mathbf{u} = \mathbf{g} & \text{in } \Omega_D. \end{cases} \tag{21}$$

We define the function spaces

$$V(\widetilde{\Omega}, \mathbb{R}^n) = \{\mathbf{u} \in L^2(\widetilde{\Omega}, \mathbb{R}^n) : \|\mathbf{u}\|_V < \infty\},$$
$$V_c(\widetilde{\Omega}, \mathbb{R}^n) = \{\mathbf{u} \in V(\widetilde{\Omega}, \mathbb{R}^n) : \mathbf{u} = 0 \text{ in } \Omega_D\},$$

where

$$\|\mathbf{u}\|_V^2 = A(\mathbf{u}, \mathbf{u}) + \|\mathbf{u}\|^2_{L^2(\widetilde{\Omega})}.$$

For given data $\mathbf{f} \in L^2(\Omega, \mathbb{R}^n)$ and $\mathbf{g} := \mathbf{v}_{|\Omega_D}$, where $\mathbf{v} \in V(\widetilde{\Omega}, \mathbb{R}^n)$, we call $\mathbf{u} \in V(\widetilde{\Omega}, \mathbb{R}^n)$ the *weak solution* to problem (21), if

$$\begin{aligned} &A(\mathbf{u}, \mathbf{v}) = (\mathbf{f}, \mathbf{v}) \ \textit{forall} \ \ \mathbf{v} \in V_c(\widetilde{\Omega}, \mathbb{R}^n), \\ &\textit{and} \ \ \mathbf{u} = \mathbf{g} \text{ in } \Omega_D. \end{aligned} \tag{22}$$

The well-posedness of problem (22) for various choices of kernel functions can be found, e.g., in [8, 21, 22, 36–38]. By exploiting the known values of $\mathbf{u}$ on the Dirichlet domain, we can rewrite the first line in (22) as

$$A_{\Omega\Omega}(\mathbf{u}, \mathbf{v}) = (\mathbf{f}, \mathbf{v}) - A_{\Omega\Omega_D}(\mathbf{g}, \mathbf{v}), \tag{23}$$

where

$$\begin{aligned} A_{\Omega\Omega}(\mathbf{u}, \mathbf{v}) :=& \int_\Omega \int_\Omega (\mathbf{v}(\mathbf{x}) - \mathbf{v}(\mathbf{y}))^\top \big(\mathbf{C}_\delta(\mathbf{x}, \mathbf{y})\mathbf{u}(\mathbf{x}) - \mathbf{C}_\delta(\mathbf{y}, \mathbf{x})\mathbf{u}(\mathbf{y})\big) d\mathbf{y} d\mathbf{x} \\ &+ 2 \int_\Omega \int_{\Omega_D} \mathbf{v}(\mathbf{x})^\top \mathbf{C}_\delta(\mathbf{x}, \mathbf{y})\mathbf{u}(\mathbf{x}) d\mathbf{y} d\mathbf{x} \end{aligned}$$

and

$$A_{\Omega\Omega_D}(\mathbf{g}, \mathbf{v}) := -2 \int_\Omega \int_{\Omega_D} \mathbf{v}(\mathbf{x})^\top \mathbf{C}_\delta(\mathbf{y}, \mathbf{x})\mathbf{g}(\mathbf{y}) d\mathbf{y} d\mathbf{x}.$$

In the stiffness matrix, the splitting (23) can be naturally obtained by separating the columns corresponding to the degrees of freedom from the columns corresponding to the nodes on the boundary Ω_D.

Table 2 Convergence rates for the truncated fractional diffusion operator (26), $\delta = 0.2$, and $h \to 0$ in a continuous Galerkin ansatz space

h	dof	L2 error	Rates
1.41e-01	1.60e+01	7.19e-04	0.00e+00
7.07e-02	8.10e+01	1.65e-04	2.13e+00
3.54e-02	3.61e+02	4.09e-05	2.01e+00
1.77e-02	1.52e+03	1.01e-05	2.01e+00
8.84e-03	6.24e+03	2.41e-06	2.07e+00

Table 3 Convergence rates for the truncated fractional diffusion operator (26), $\delta = 0.2$, and $h \to 0$ in a discontinuous Galerkin ansatz space

h	dof	L2 error	Rates
1.41e-01	1.44e+02	7.58e-04	0.00e+00
7.07e-02	5.94e+02	1.60e-04	2.24e+00
3.54e-02	2.39e+03	4.06e-05	1.98e+00
1.77e-02	9.59e+03	1.01e-05	2.01e+00
8.84e-03	3.84e+04	2.43e-06	2.06e+00

3.1 Truncated Fractional-Type Diffusion

We choose the scalar-valued translationally invariant and symmetric kernel function $\gamma_\delta^s : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ given by [22]

$$\gamma_\delta^s(\mathbf{x}, \mathbf{y}) = c_{s,\delta} \, \frac{1}{|\mathbf{x} - \mathbf{y}|^{2+2s}}, \tag{24}$$

where

$$c_{s,\delta} = \frac{2-2s}{\pi \delta^{2-2s}} \quad \text{and} \quad s \in (0,1). \tag{25}$$

Then, the scalar-valued truncated fractional-type diffusion operator reads as

$$-\mathcal{L}_\delta^s u(\mathbf{x}) := \int_{B_\delta(\mathbf{x})} \gamma_\delta^s(\mathbf{x}, \mathbf{y})(u(\mathbf{x}) - u(\mathbf{y}))d\mathbf{y}. \tag{26}$$

The well-posedness of problem (22) for this choice of kernel is studied in [22]. The constant c_δ depends on δ and s and is chosen such that the operator converges to the classical Laplacian as $\delta \to 0$; see, e.g., [26, Lemma 7.4.1]. Another choice of the constant convergence to the fractional Laplacian as $\delta \to \infty$ can also be obtained [39].

In the example above, we choose the manufactured solution $u(\mathbf{x}) = x_1^2 x_2 + x_2^2$ and set $f(\mathbf{x}) := -\Delta u(\mathbf{x}) = -2(x_2 + 1)$ in Ω and $g(\mathbf{x}) := u(\mathbf{x})$ on Ω_D. Since the correctly scaled nonlocal operator equals the classical Laplacian operator on polynomials of order up to three (see, e.g., [26]), we have that $u(\mathbf{x})$ is the solution of problem (22). Furthermore, we choose $s := 0.5$, $\delta = 0.2$, and various mesh sizes h as given in the tables below. In view of Table 1, for pairs of disjoint elements, we use as $\heartsuit$ a 7-point quadrature rule[6] for each, i.e., outer and inner, integral. Since six of the seven points are located on the boundary of the triangle, this choice has proven to be advantageous in the case of truncated kernels since the resulting interaction neighborhoods centered at these points better approximate the interaction domain of this triangle; also, see Fig. 3, and for more details, see [1]. For intersecting pairs, we need a quadrature rule on $(0, 1)^4$ after the integral transformations are performed. For $\clubsuit$, we choose a tensor product of a 5-point Gauss quadrature rule, which is sufficiently accurate to preserve the expected convergence rates.

The convergence rates on a continuous Galerkin ansatz space are shown in Table 2. For our choice of $s = 0.5$, a discontinuous Galerkin ansatz space is also conforming, and the

[6] Specifically, the quadrature points are the barycenter, the vertices, and the midpoints of the sides of the reference triangle $\widehat{\mathcal{E}}$ and the corresponding weights are $\frac{27}{60} \cdot \frac{1}{2}$, $\frac{3}{60} \cdot \frac{1}{2}$, and $\frac{8}{60} \cdot \frac{1}{2}$, respectively.

Table 4 Convergence rates for the peridynamics operator (29), $\delta = 0.1$, and $h \to 0$ in a continuous Galerkin ansatz space

h	dof	L2 error	Rates
1.41e-01	3.20e+01	7.47e-04	0.00e+00
7.07e-02	1.62e+02	1.82e-04	2.04e+00
3.54e-02	7.22e+02	4.58e-05	1.99e+00
1.77e-02	3.04e+03	1.12e-05	2.04e+00
8.84e-03	1.25e+04	2.63e-06	2.09e+00

results are presented in Table 3. In both settings, we observe the expected second-order convergence as the mesh size $h \to 0$; see, e.g., [22]. Note that the given examples violate Assumption 4 in the first stage of the experiments as $2h > \delta$. We see that the first rates in both tables are affected by this.

3.2 Bond-Based Peridynamics

The translationally invariant and symmetric *linear peridynamic* kernel is given by [38]

$$\mathbf{C}_\delta(\mathbf{x},\mathbf{y}) := c_\delta\, \mathbf{C}(\mathbf{x}-\mathbf{y})\, \mathbb{1}_{B_\delta(\mathbf{x})}(\mathbf{y}), \tag{27}$$

where

$$\mathbf{C}(\mathbf{x}-\mathbf{y}) := \frac{(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^\top}{|\mathbf{x}-\mathbf{y}|^3} \text{ and } c_\delta := \frac{3}{\delta^3}. \tag{28}$$

The corresponding linear peridynamics operator then reads as

$$-\mathcal{P}_\delta \mathbf{u}(\mathbf{x}) := c_\delta \int_{B_\delta(\mathbf{x})} C(\mathbf{x}-\mathbf{y})(\mathbf{u}(\mathbf{x}) - \mathbf{u}(\mathbf{y}))d\mathbf{y}. \tag{29}$$

In [38], the well-posedness is established for problem (22), where $\mathbf{f} \in L^2(\Omega, \mathbb{R}^d)$ and $\mathbf{g} \in L^2(\Omega_D, \mathbb{R}^d)$. For the given constant in (28), it is also shown there that the peridynamics operator $-\mathcal{P}_\delta$ converges to the local Navier operator

$$-\mathcal{P}_0 \mathbf{u}(\mathbf{x}) := -\frac{\pi}{4}\Delta \mathbf{u}(\mathbf{x}) - \frac{\pi}{2}\nabla \operatorname{div} \mathbf{u}(\mathbf{x}) \tag{30}$$

as $\delta \to 0$. A similar convergence result is also obtained for the corresponding weak solutions. Thus, in the given example, we choose the manufactured polynomial solution $\mathbf{u}(\mathbf{x}) := (x_2^2, x_1^2 x_2)$, set $\mathbf{f}(\mathbf{x}) := -\mathcal{P}_0 \mathbf{u}(\mathbf{x}) = -\frac{\pi}{2}\left(1 + 2x_1, x_2\right)$ in Ω and as Dirichlet constraints choose $\mathbf{g}(\mathbf{x}) := \mathbf{u}(\mathbf{x})$ on Ω_D. Similarly to the diffusion case above, we again obtain that $\mathbf{u}(\mathbf{x})$ is the solution of (22) due to the correct scaling of the operator [38]. The results are

Table 5 Convergence rates for peridynamics operator (29), $\delta = 0.1$, and $h \to 0$ in a discontinuous Galerkin ansatz space

h	dof	L2 error	Rates
1.41e-01	2.88e+02	6.64e-04	0.00e+00
7.07e-02	1.19e+03	1.86e-04	1.84e+00
3.54e-02	4.78e+03	4.96e-05	1.90e+00
1.77e-02	1.92e+04	1.25e-05	1.99e+00
8.84e-03	7.68e+04	2.92e-06	2.10e+00

Table 6 Convergence rates and timings for the infinity normball (31), $\delta = \sqrt{2}h$, and $h, \delta \to 0$ in a continuous Galerkin ansatz space

h	δ	L2 error	Rates	Time (s)
1.41e-01	2.00e-01	2.00e-01	0.00e+00	2.98e-02
7.07e-02	1.00e-01	4.01e-02	2.32e+00	1.45e-01
3.54e-02	5.00e-02	8.85e-03	2.18e+00	4.82e-01
1.77e-02	2.50e-02	2.10e-03	2.07e+00	1.86e+00
8.84e-03	1.25e-02	5.17e-04	2.03e+00	7.55e+00

presented in Tables 4 and 5 for a continuous and discontinuous Galerkin ansatz, respectively. In both settings we observe second-order convergence as the mesh size $h \to 0$.

3.3 Diffusion with Infinity Ball Truncation

Here, we consider a constant kernel truncated by the infinity normball. Specifically, we choose the constant to be $c_\delta^\infty := \frac{3}{4\delta^4}$, which ensures the convergence to the local Dirichlet problem for vanishing horizon $\delta \to 0$; see, e.g., [26]. The nonlocal operator is then given by

$$-\mathcal{L}_\delta^\infty u(\mathbf{x}) := c_\delta^\infty \int_{B_\delta^\infty(\mathbf{x})} (u(\mathbf{x}) - u(\mathbf{y}))d\mathbf{y}. \tag{31}$$

The truncation by the infinity normball is implemented without geometric error, which allows numerical tests of the asymptotic compatibility of the finite element discretization [30]. For the numerical experiment, we choose the manufactured solution $u(\mathbf{x}) = \sin(4\pi x_1)\sin(4\pi x_2)$, set $f(\mathbf{x}) := -\Delta u(\mathbf{x}) = 32\pi^2 \sin(4\pi x_1)\sin(4\pi x_2)$ in Ω and $g(\mathbf{x}) := u(\mathbf{x})$ on Ω_D. Note that, opposed to the previous examples, here, for a fixed $\delta > 0$, the function $-\mathcal{L}_\delta^\infty u$ differs from $-\Delta u$. Consequently, the solutions of the nonlocal and the local Dirichlet problem differ from each other for each $\delta > 0$ and coincide in the limit $\delta \to 0$. In fact, the solution to (31) changes as δ changes, and we can observe the convergence in Table 6. We run tests for a fixed mesh size h and vanishing δ (see Table 7), as well as for a horizon-dependent mesh size $h = \sqrt{2}\delta$ and vanishing δ (see Table 6). In both cases, we observe a second-order convergence as $\delta \to 0$.

3.4 Nonlocal Convection-Diffusion

Nonsymmetric kernels can model convective effects, and nlfem allows to assemble the respective systems. In this example, we consider a convection-diffusion operator

Table 7 Convergence rates for the infinity normball (31), fixed h, and $\delta \to 0$ in a continuous Galerkin ansatz space

h	δ	L2 error	Rates
8.84e-03	2.00e-01	2.03e-01	0.00e+00
8.84e-03	1.00e-01	3.98e-02	2.35e+00
8.84e-03	5.00e-02	8.81e-03	2.18e+00
8.84e-03	2.50e-02	2.08e-03	2.08e+00
8.84e-03	1.25e-02	5.17e-04	2.01e+00

$$-\mathcal{L}_{\delta}^{cd}u(\mathbf{x}) = 2\int_{\widetilde{\Omega}} \vartheta(\mathbf{x},\mathbf{y})u(\mathbf{x}) - \vartheta(\mathbf{y},\mathbf{x})u(\mathbf{y})d\mathbf{y}, \tag{32}$$

with $\vartheta = \vartheta^d + \vartheta^c$ consisting of a constant diffusion term

$$\vartheta^d(\mathbf{x},\mathbf{y}) = \epsilon\ \mathbb{1}_{B_{\delta}^{\infty}(\mathbf{x})}(\mathbf{y})\frac{3}{4\delta^4}, \tag{33}$$

and an antisymmetric kernel

$$\vartheta^c(\mathbf{x},\mathbf{y}) = \mathbb{1}_{B_{\delta}^{\infty}(\mathbf{x})}(\mathbf{y})\frac{\mathbf{b}^T(\mathbf{y}-\mathbf{x})}{2}\frac{3}{4\delta^4}, \tag{34}$$

with $\mathbf{b} = (1,1)^T$, to account for the convective effects. The factor $\epsilon > 0$ controls the influence of the diffusion in the model, and it is set to 10^{-3}. In order to show the convergence of the approximation, we choose the solution $u(\mathbf{x}) = x_1^2 + x_2^2$ for which we obtain $-\mathcal{L}_{\delta}^{cd}u(\mathbf{x}) = -4\epsilon + 2x_1 + 2x_2 =: f(\mathbf{x})$. Note that the manufactured solution in fact fulfills

$$-\mathcal{L}_{\delta}^{cd}u(\mathbf{x}) = -\epsilon\ \Delta u(\mathbf{x}) + \mathbf{b}^T\nabla u(\mathbf{x}). \tag{35}$$

The above differential operator is, in fact, a special case of a local convection-diffusion operator because the vector field $\mathbf{b}$ is constant in $\mathbf{x}$ and thus has zero divergence. The second-order convergence is shown in Table 8. Another example with $\epsilon = 0.1$ is plotted in Fig. 5.

The code nlfem only allows to assemble stiffness matrices for kernels with sufficiently large diffusive part ϑ^d, and the convergence of the convection-diffusion problem becomes unstable for $\epsilon = 10^{-5}$. The modeling of stronger convective effects requires upwind integration schemes like the one presented in [40].

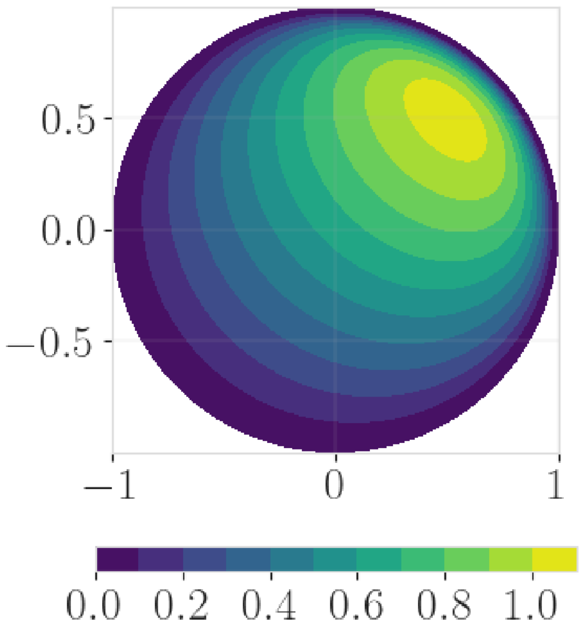

Fig. 5 Contour plot of the solution to a nonlocal convection-diffusion problem with constant forcing term $f(x) \equiv 1$, Dirichlet zero boundary conditions, horizon $\delta = 0.01$, and kernel ϑ with $\epsilon = 0.1$

Table 8 Convergence of the nonlocal convection-diffusion problem for $\delta = 0.1$ and $h \to 0$

h	dof	$L2$ error	Rates
1.41e-01	1.60e+01	1.28e-03	0.0e+00
7.07e-02	8.10e+01	3.36e-04	1.93e+00
3.54e-02	3.61e+02	8.39e-05	2.00e+00
1.77e-02	1.52e+03	2.09e-05	2.01e+00
8.84e-03	6.24e+03	5.11e-06	2.03e+00

3.5 Parallel Complexity of the Assembly Process

The number of elements in each interaction neighborhood grows quadratically in 2d if the diameter of the elements h is decreased for fixed δ. Thus, the assembly of the system matrix with retriangulations as described in the beginning of Sect. 2.4.1 becomes a costly procedure. Therefore, a matrix-free approach is too expensive, and we store the stiffness matrix in a sparse format. Furthermore, it makes sense to share the work among multiple threads. The multithreading is implemented using OpenMP [41], and the work is shared by a partitioning of finite elements as given in Algorithm 2, line 1. Due to the nonlocality of the operator, several threads might need to access identical entries in the global stiffness matrix at the same time to store their contribution. OpenMP allows so-called critical sections to organize the manipulation of shared variables. This avoids write conflicts, but it would tremendously slow down the computation. In order to avoid a critical section during the assembly, each thread separately allocates its portion of the global stiffness matrix. The size of the overlap among the submatrices in the threads, i.e., the amount of additional memory requirements due to the parallelization, depends on the nonlocal overlap of the subdomains. We can therefore reduce the memory requirements by partitioning the domain with metis [42] instead of a scheduler of OpenMP. The submatrices are finally added together into a single sparse matrix.

We depict the strong scaling on a computer with two 2.20GHz Intel Xeon CPUs with 22 cores per socket and two threads per core. The machine has 756 GB of RAM. Table 9

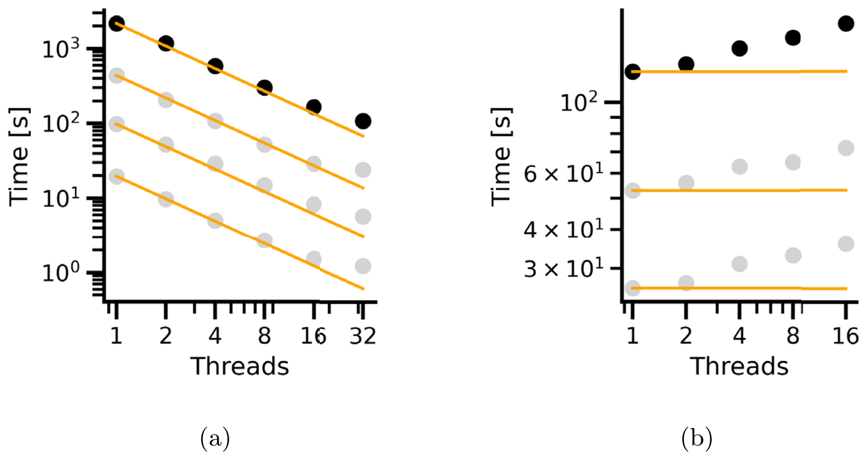

Fig. 6 **a** Strong scaling for 24,336 (black) and 10,848, 6120, and 2736 (gray) degrees of freedom. **b** Weak scaling for horizons 0.06, 0.05, and 0.04 (black)

Table 9 Assembly time for a system with 24,336 degrees of freedom in strong scaling study

Threads	1	2	4	8	16	32
Time (s)	2153	1169	583	302	166	107
Parallel efficiency	-	0.92	0.92	0.89	0.81	0.63

and Fig. 6a (black dots) show the run time of an assembly on a regular grid on a domain $\widetilde{\Omega} = [-\delta, 0.5 + \delta]^2$ with mesh size $h = 7.1\text{e}{-}03$ and $\delta = 0.1$. The related linear system has 24,336 degrees of freedom and 45,022,167 nonzero entries. The number of threads is increased by a factor up to 32 while the time drops by a factor of 1/20. The scaling looks perfect for up to 16 parallel threads and the effect diminishes from then on. Figure 6a also shows the scaling for smaller problems with 10,848, 6120, and 2736 degrees of freedom (gray dots) which show a similar behavior.

The weak scaling experiment has been performed on a machine with four 2.1GHz AMD Opteron 6272 Processors with 8 cores per socket and 2 threads per core. The domain $\widetilde{\Omega} = [-\delta, \sqrt{T} + \delta]^2$ depends on the number of threads $T = 1, 2, \ldots, 16$. For the study, we choose horizons $\delta = 0.06, 0.05$, and 0.04 and set the mesh size to $\delta = 3h$. The timings for the largest experiment with $\delta = 0.04$ (black dots, Fig. 6b) are depicted in Table 10. The system size grows linearly in the number of cores (dof, Table 10). Therefore, a perfect weak scaling should exhibit no increase in the computational time for growing number of threads so that the *scaled speedup* is equal to the number of threads. Figure 6b depicts that the assembly does not scale perfectly. More precisely, Table 10 shows that the 16 threads yield a speedup of 11. This can be explained by a growing overhead in the merging of the submatrices into a single sparse system.

However, the increase in precision for vanishing mesh size is quadratic which can amortize the costs. This is reported in Table 6 where we find that an increase of computation time by a factor 253 leads to a 386 times smaller L^2 error.

4 The Code nlfem

4.1 Usage

The package provides an assembly routine while clearly the construction of numerical examples also incorporates the definition of a finite element mesh, the settings of the integrators, kernels and forcing terms, and finally the solution of the resulting equation. We exemplify the usage of nlfem by solving a scalar, nonlocal Dirichlet-type problem (see (21)), where Ω is given by a 2d-disk of radius 0.9 with a nonlocal boundary Γ^D of width $\delta = 0.1$.

Table 10 Assembly time for the system with horizon $\delta = 0.04$ degrees of freedom in weak scaling study

Threads	1	2	4	8	16
Time (s)	125	132	148	160	170
Scaled speedup	1	1.89	3.38	6.25	11.3
dof	5625	11,025	22,201	44,100	88,209

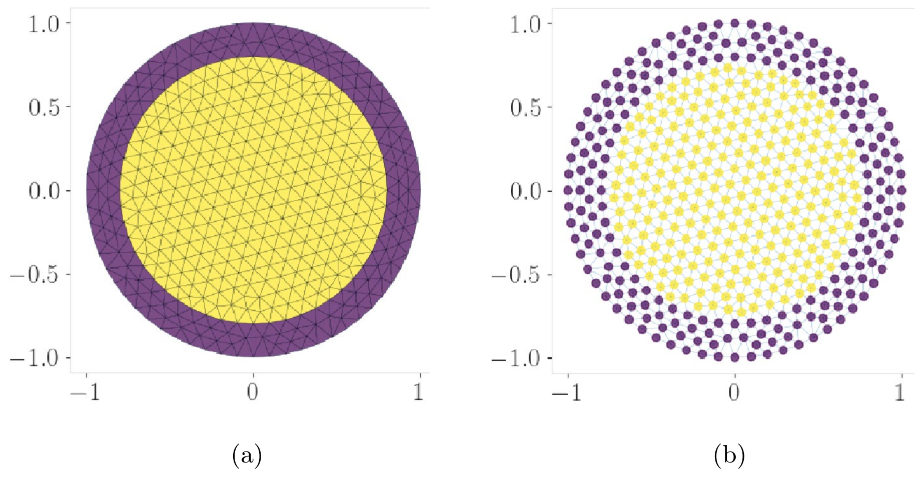

Fig. 7 **a** The domain $\widetilde{\Omega}$ consisting of Ω (yellow) and Ω_D (blue). **b** The degrees of freedom with the corresponding labels which determine whether a degree of freedom is unknown (yellow) or given by the Dirichlet data (blue)

4.1.1 Finite Element Mesh

The construction requires a finite element mesh in a 1d, 2d, or 3d domain. The mesh is characterized by its `elements` and `vertices`. The `elements` are given by a `numpy.ndarray` of datatype `numpy.int_` and shape `(nE, d+1)`, where `nE` is the number of elements and `d` the dimension of the domain. The `vertices` are a numpy array of floats with shape `(nV, d)`, where `nV` is the number of vertices.

The domain described by the above arrays needs to be divided into different parts according to their purpose. To that end, we define a numpy `nd.array` called `ele-mentLabels` of type `numpy.int_` and length `nE`. It assigns a label to each element. Negative elements indicate the Dirichlet boundary, any positive element is considered a part of the domain, and zero-labeled elements are ignored and used to manipulate the mesh topology only; see Assumption 4 and the related remarks.

Example The finite element mesh (that is `vertices` and `elements`) of the required form can be obtained for example from gmsh. In the example presented in Fig. 7a, the `elementLabels` are set to -1 on Γ^D (blue in the below figure) and 1 on Ω (yellow) in Fig. 7a.

4.1.2 Defining the Settings

The `kernel` is a Python dictionary which contains the keys `function`, `hori-zon`, `outputdim`, and possibly `fractional_s`. A list of implemented kernel functions can be found via `nlfem.show_options()`. The `horizon` determines the interaction radius of the kernel and `outputdim` tells whether the kernel is scalar or tensor-valued. Given the kernel has a singularity which requires regularizing integral transformations, we need to specify the parameter s of the singularity in `fractional_s`. The quantity is required for the special integration routines `fractional` and `weakFractional` which need to be applied to those kernels.

Example We choose the scalar kernel (24) which is of truncated fractional type where $d = 2$ and $s = 0.4$. The corresponding dictionary reads as

```
kernel = {
    "function": "fractional",
    "horizon": 0.1,
    "outputdim": 1,
    "fractional_s": 0.4
}
```

As optional parameter, it is possible to hand over varying kernel coefficients with the key `"Theta"`. Of course, the implemented kernel has to support the usage of the given information. This is so, for example, for the kernel functions `"theta"` and `"sparsetheta"` which evaluate

$$\frac{4}{\pi\delta^4}\Theta(\mathbf{x},\mathbf{y}) \text{ and } \frac{4}{\pi\delta^4}(1-\Theta(\mathbf{x},\mathbf{y})),$$

respectively. The coefficients are expected to be of type `scipy.sparse.csr_matrix`. However, there are no other restrictions, so that any use of the coefficients inside of the kernel function is possible. Note also that new kernels can be defined.

The dictionary specifying the `forcing`-term has a similar structure and contains a key `function`. Again, a list of implemented forcing functions can be found via `nlfem.show_options()`. Note that the assembly of the right-hand side is standard and nlfem offers this functionality for convenience only.

Example We choose $f(\mathbf{x}) = -2(x_1 + 1)$ which can be specified by

```
forcing = {"function": "linear"}
```

All other settings are stored in the dictionary `conf`. The truncation routines are selected by the `method` given in `approxBalls`. The quadrature rule which is used in the truncation routine is indicated by numpy arrays of quadrature points and weights. For kernels which exhibit a singularity, it is possible to select another integration routine specifically for touching elements (e.g., `fractional`, `weakFractional`). However, the choice does not affect the quadrature for nonsingular kernels. The function `nlfem.show_options()` prints a list of the singular kernels.

Example We discretize the problem with a continuous Galerkin ansatz space and use a retriangulation with caps (Definition 2.3) to approximate the truncation of the interaction neighborhood. We choose a 7-point quadrature rule with points `Px` and weights `dx`. The full example configuration is specified by the dictionary

```
conf = {
    "ansatz": "CG",
    "approxBalls": {
        "method": "retriangulate"
    },
    "quadrature": {
        "outer": {
            "points": Px,
    },      "weights": dx
    "inner": {
        "points": Px,
        "weights": dx
    },
    "touchingElements": {
        "method": "fractional",
        "ntensorGaussPoints": 4
    }
},
"verbose": True
}
```

Note that all options for the settings in `kernel`, `function`, and `conf` can be printed by `nlfem.show_options()` and empty dictionaries in the required form can be obtained from `nlfem.get_empty_settings()`.

4.1.3 Assembly

Based on the settings, the assembly of the nonlocal stiffness matrix is effected via

```
import nlfem
mesh, A = nlfem.stiffnessMatrix_fromArray(elements,
elementLabels, vertices, kernel, conf)
f_ = nlfem.loadVector(mesh, forcing, conf)
```

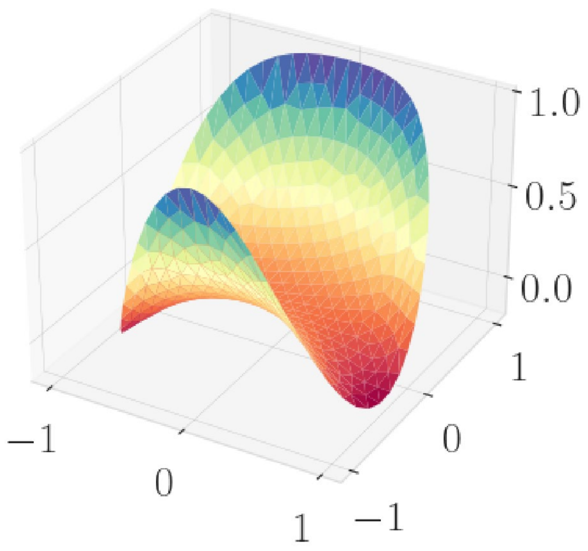

Fig. 8 Solution of the example problem

The dictionary `mesh` contains information about the labeling of `elements`, `vertices`, and `dof`. The labels of the degrees of freedom (dof) depend on the kernel (scalar or tensor-valued) and the ansatz space (CG or DG). Therefore, the function `nlfem.stiffnessMatrix_fromArray()` automatically deduces `vertexLabels` and `dofLabels`. A vertex is labeled according to the labels of the elements it belongs to, and it is given the smallest of those element labels. The dof labels are identical to the vertex labels for CG ansatz spaces and scalar-valued kernels. For DG ansatz spaces, the labels are directly obtained from the element labels. The labels of the degrees of freedom on the domain Ω and the Dirichlet domain Ω_D are derived from the element labels and stored in `mesh` as `dofLabels` so that the following lines isolate the corresponding entries of the discrete solution (Fig. 8).

```
Omega = mesh["dofLabels"] > 0
Gamma = mesh["dofLabels"] < 0
```

Example In the given example, we have a CG ansatz space and a scalar-valued kernel so that the `dofLabels` are identical to the `vertexLabels`; see Fig. 7b.

The solution of a Dirichlet-type problem requires the definition of Dirichlet data g. The function below implements $g(\mathbf{x}) = x_1^2 x_2 + x_2^2$.

```
def g(vertices):
    return vertices[:, 0]**2*vertices[:, 1] + vertices[:, 1]**2
```

Ultimately, we can solve the discrete nonlocal Dirichlet problem via

```
from scipy.sparse.linalg import cg
f = f_[Omega]
A_OO = A[Omega][:, Omega]
A_OD = A[Omega][:, Gamma]
g_D = g(vertices[Gamma])
f = f - A_OD.dot(g_D)

u = numpy.zeros(vertices.shape[0], dtype=float)
u[Omega] = cg(A_OO, f)[0]
u[Gamma] = g_D
```

4.2 Structure of the Code

The nlfem code provides a Python interface which communicates all settings to a C++ function. The main functionality of nlfem is evoked by the function `stiffnessMatrix_fromArray()` in the file `cython/nflem.pyx`. We therefore restrict the description of the code structure on the flow of function calls starting from the user input to the evaluation of a kernel function and the return of the discrete system. More details are to be found in the C++ documentation of nlfem. The function `stiffnessMatrix_fromArray()` is written in Cython and can be called from Python. It passes the input to a C++ function and leads to a call of `par_system()` located in `src/Cassemble.cpp`. This function collects all settings and starts the assembly of the nonlocal stiffness matrix. It splits up the work by an OpenMP work-sharing construct and starts a double loop over the finite elements for each of the workers. In the center of the double loop, `par_system()` calls the integration function which has been chosen by the user. The function is called by the pointer `integrate()` and implemented in `src/integration.cpp`. This function again evokes a specific method to evaluate the interaction neighborhood which finally evaluates the kernel `model_kernel()`. The kernels are implemented in `src/model.cpp` and listed in a C++ map in `src/Cassemble.cpp` which allows to access them from the Python interface. After the completion of the assembly, a sparse matrix is stored to disk by `par_system()` and read again by `stiffnessMatrix_fromArray()` which returns it to the user as `scipy.sparse.csr_matrix` object.

4.3 Scope and Comparison

The code nlfem [10] provides similar functionality as PyNucleus [19], and we therefore combine the presentation of the scope of nlfem with a comparison of the functionality of PyNucleus and nlfem. First of all, the code PyNucleus points into a different direction than nlfem as it assembles operators of the type

$$-\widetilde{\mathcal{L}}\mathbf{u}(\mathbf{x}) = \int_{\mathbb{R}^d} (u(\mathbf{x}) - u(\mathbf{y}))\gamma(\mathbf{x},\mathbf{y})d\mathbf{y} \tag{36}$$

as opposed to (4), where the case that the scalar kernel $\gamma(\mathbf{x}, \mathbf{y})$ has an infinite interaction horizon is explicitly allowed. The operators $\mathcal{L}$ and $\widetilde{\mathcal{L}}$ are identical for symmetric kernels.

Table 11 Comparison of functionality

	PyNucleus	nlfem
Interface	Python	Python
Parallelization	MPI	OpenMP
Mesh construction	✓	✗
Data format	Dense, hierarchical, sparse	Sparse
Solvers	✓	✗
Kernel domain	1d, 2d	1d, 2d, 3d
Kernel value	Scalar	Tensor, scalar
Varying coefficients	✓	✓
Varying fractional order	✓	✗

We show an overview of the functionality of the two codes in Table 11. PyNucleus aims to provide efficient discretization and assembly routines with quasi-optimal complexity, in particular for problem with infinite interaction horizon δ, which nlfem does not cover. Both codes provide a Python interface and implement some kind of parallelization, where PyNucleus can directly be used on clusters (MPI), as opposed to nlfem which offers multi-threading (OpenMP). PyNucleus contains mesh construction and solver functionality, and it can store the matrices resulting from the assembly in hierarchical, dense, and sparse format, where nlfem offers sparse matrices only. Both codes provide discontinuous and continuous finite element spaces on 1d, 2d, and 3d domains. PyNucleus allows discontinuous P0 and continuous P1, P2, and P3 elements while nlfem provides discontinuous and continuous P1 elements. While PyNucleus is restricted to scalar kernels, nlfem allows scalar- and tensor-valued kernels.

Remark 4.1 While tensor-valued kernels allow systems related to linearized peridynamics models, nlfem does not allow to approximate nonlinear peridynamics operators.

In Table 12, we see a detailed overview of the truncations and kernel types which are implemented. While in both codes new kernels can be introduced, adding new truncation routines or special quadrature rules is a more complex endeavor. We therefore compare the scope of the codes with respect to the implemented interaction horizon and singularities.

Table 12 Comparison of truncation and kernels

			PyNucleus	nlfem
Dimension	Truncation	Singularity		
1d	$\delta = \infty$	Fractional	✓	✗
	$\delta < \infty$	Integrable	✓	✓
		Fractional	✓	✗
2d	$\delta = \infty$	Fractional	✓	✗
	ℓ_∞	Fractional, integrable	✗	✓
	approxcaps	Fractional, integrable	✓	✓
1d, 2d, 3d	*barycenter*	Integrable	✗	✓

On 1d domains, PyNucleus can assemble fractional and integrable kernels with finite and infinite horizons, where nlfem supports truncated integrable kernels only. On 2d domains, both codes support fractional-type and integrable kernels.

Remark 4.2 The quadrature rules for the (truncated) fractional Laplacian, that is for $\mathbf{\Psi}_\delta(\mathbf{x}, \mathbf{y}) \equiv const$ in (1), can be simplified significantly [18]. Identical elements require an evaluation of a 1d instead of a 4d integral. Similarly, edge-touching and vertex-touching elements require the computation of two- and three-dimensional integrals only [18]. As those rules have been incorporated into PyNucleus, it is to be preferred over nlfem for the truncated fractional Laplacian. Note however that the rules in nlfem are directly applicable to a larger range of kernels as, for example, the tensor-valued nonradial kernel (27).

Both codes offer error commensurate approximations of B_δ^2 truncations. The code nlfem moreover supports the B_δ^∞ and *barycenter* ball approximations [1]. The main advantage of the latter is that it can be implemented independently of the dimension d of the domain. Therefore, nlfem can assemble nonlocal operators in 3d. On the other hand, PyNucleus contains assembly routines for classical, local operators, which nlfem does not support.

To conclude, we find that PyNucleus covers a larger scope than nlfem[,] while not all features of nlfem are contained in PyNucleus .

5 Conclusion

The code nlfem is a tool to set up numerical experiments for researchers. The documentat ion also describes the extension by user-defined kernels which allows to consider a large problem class. It can assemble nonlocal problems in 2d with an error commensurate kernel truncations for Euclidean and infinity norm balls and in 1d and 3d using the generically implemented barycenter method [1]. We therefore hope to bridge efficiency and flexibility to obtain a convenient Python package which nourishes the current development in the field of finite element methods for nonlocal operators and enables easy validations of new theory without the effort of implementing code from scratch. In that sense, nlfem contributes to the ongoing effort to unlock the full potential of nonlocal models.

Author Contribution M.K. is the main developer of the code. Together with C.V., he wrote the manuscript. V.S. reviewed and advised.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability The code nlfem is published under the GNU General Public License version 3 and can be freely downloaded at https://gitlab.uni-trier.de/pde-opt/nonlocal-models/nlfem.

Declarations

Ethics Approval Not applicable

Conflict of Interest The authors declare no competing interests.

References

1. D'Elia M, Gunzburger M, Vollmann C (2021) A cookbook for approximating Euclidean balls and for quadrature rules in finite element methods for nonlocal problems. Math Models Methods Appl Sci 31(08):1505–1567
2. Dahal B, Seleson P, Trageser J (2022) The evolution of the peridynamics co-authorship network. J Peridyn Nonlocal Model 1–45
3. Brockmann D (2008) Anomalous diffusion and the structure of human transportation networks. Eur Phys J Special Topics 157(1):173–189
4. Brockmann D, Theis F (2008) Money circulation, trackable items, and the emergence of universal human mobility patterns. IEEE Pervasive Comput 7(4):28–35
5. Silling SA (2000) Reformulation of elasticity theory for discontinuities and long-range forces. J Mech Phys Solids 48(1):175–209
6. Gilboa G, Osher S (2009) Nonlocal operators with applications to image processing. Multiscale Model Simul 7(3):1005–1028
7. Peyré G, Bougleux S, Cohen L (2008) Non-local regularization of inverse problems. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 57–68
8. Du Q, Gunzburger M, Lehoucq RB, Zhou K (2013) A nonlocal vector calculus, nonlocal volume-constrained problems, and nonlocal balance laws. Math Models Methods Appl Sci 23(03):493–540
9. Bogdan K, Burdzy K, Chen Z-Q (2003) Censored stable processes. Probab Theory Relat Fields 127(1):89–152
10. Klar M, Vollmann C. The nlfem project website. https://gitlab.uni-trier.de/pde-opt/nonlocal-models/nlfem. Accessed 3 Jul 2023
11. Boys B, Dodwell TJ, Hobbs M, Girolami M (2021) PeriPy-a high performance OpenCL peridynamics package. Comput Methods Appl Mech Eng 386
12. Littlewood DJ, Parks ML, Foster JT, Mitchell JA, Diehl P (2023) The peridigm meshfree peridynamics code. J Peridyn Nonlocal Model 11:1–31
13. Parks ML, Seleson P, Plimpton SJ, Silling SA, Lehoucq RB (2011) Peridynamics with LAMMPS: a user guide, v0. 3 beta. Sandia Rep (2011–8253) 3532
14. Jha PK, Diehl P (2021) NLMech: implementation of finite difference/meshfree discretization of nonlocal fracture models. J Open Source Softw 6(65):3020
15. Leng Y, Tian X, Trask N, Foster JT (2021) Asymptotically compatible reproducing kernel collocation and meshfree integration for nonlocal diffusion. SIAM J Numer Anal 59(1):88–118
16. Sauter SA, Schwab C (2011) Boundary element methods, vol 39. Springer Series in Computational Mathematics, Springer, Berlin Heidelberg
17. Acosta G, Bersetche FM, Borthagaray JP (2017) A short FE implementation for a 2D homogeneous Dirichlet problem of a fractional Laplacian. Comput Math Appl 74(4):784–816
18. Ainsworth M, Glusa C (2018) Towards an efficient finite element method for the integral fractional Laplacian on polygonal domains. In: Contemporary Computational Mathematics - A Celebration of the 80th Birthday of Ian Sloan, pp 17–57
19. Glusa C. The Pynucleus Project website. https://github.com/sandialabs/PyNucleus. Accessed 3 Jul 2023
20. Ren B, Wu CT, Askari E (2017) A 3D discontinuous Galerkin finite element method with the bond-based peridynamics model for dynamic brittle failure analysis. Int J Impact Eng 99:14–25
21. D'Elia M, Du Q, Glusa C, Tian X, Zhou Z (2020) Numerical methods for nonlocal and fractional models. Acta Numerica 29
22. Du Q, Gunzburger M, Lehoucq RB, Zhou K (2012) Analysis and approximation of nonlocal diffusion problems with volume constraints. SIAM Rev 54(4):667–696
23. Dipierro S, Ros-Oton X, Valdinoci E (2017) Nonlocal problems with Neumann boundary conditions. Revista Matematica Iberoamericana 33(2):377–416

24. Foghem G, Kassmann M (2022) A general framework for nonlocal Neumann problems. https://arxiv.org/abs/2204.06793
25. Schuster M, Schulz V, Vollmann C (2022) Shape optimization for interface identification in nonlocal models. arXiv:1909.08884
26. Vollmann C (2019) Nonlocal models with truncated interaction Kernels - analysis, finite element methods and shape optimization. Doctoral Thesis, Universität Trier
27. Bobaru F, Hu W (2012) The meaning, selection, and use of the peridynamic horizon and its relation to crack branching in brittle materials. Int J Fract 176:215–222
28. Parks M, Lehoucq R, Plimpton S, Silling S (2008) Implementing peridynamics within a molecular dynamics code. Comput Phys Commun 179(11):777–783
29. Bondy A, Murty MR (2008) Graph theory. Graduate texts in mathematics. Springer-Verlag
30. Tian X, Du Q (2014) Asymptotically compatible schemes and applications to robust discretization of nonlocal models. SIAM J Numer Anal 52(4):1641–1665
31. Tao Y, Tian X, Du Q (2017) Nonlocal diffusion and peridynamic models with Neumann type constraints and their numerical approximations. Appl Math Comput 305:282–298
32. Gunzburger M, Lehoucq RB (2010) A nonlocal vector calculus with application to nonlocal boundary value problems. Multiscale Model Simul 8(5):1581–1598
33. Yin X, Du Q, Xie H (2022) On the convergence to local limit of nonlocal models with approximated interaction neighborhoods. SIAM J Numer Anal 60(4):2046–2068
34. Da Fies G, Vianello M (2012) Algebraic cubature on planar lenses and bubbles. Dolomites Res Notes Approx 5:7–12
35. Zhou K, Du Q (2010) Mathematical and numerical analysis of linear peridynamic models with nonlocal boundary conditions. SIAM J Numer Anal 48(5):1759–1780
36. Du Q (2019) Nonlocal modeling, analysis, and computation. SIAM
37. Du Q, Gunzburger M, Lehoucq RB, Zhou K (2012) Analysis and approximation of nonlocal diffusion problems with volume constraints. SIAM Rev 54(4):667–696
38. Mengesha T, Qiang D (2014) The bond-based peridynamic system with Dirichlet-type volume constraint. Proc Roy Soc Edinburgh Sect A 144(1):161–186
39. D'Elia M, Gunzburger M (2013) The fractional Laplacian operator on bounded domains as a special case of the nonlocal diffusion operator. Comput Math Appl 66:1245–1260
40. Tian H, Ju L, Du Q (2015) Nonlocal convection-diffusion problems and finite element approximations. Comput Methods Appl Mech Eng 289:60–78
41. Dagum L, Menon R (1998) OpenMP: an industry standard API for shared-memory programming. IEEE Comput Sci Eng 5(1):46–55
42. Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 20(1):359–392