PDMATLAB2D: A Peridynamics MATLAB Two-Dimensional Code

Pablo Seleson[1*], Marco Pasetto[2†], Yohan John[3,4†], Jeremy Trageser[1,5] and Samuel Temple Reeve[6]

[1]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, MS-6013, Oak Ridge, TN, 37831-6013, United States[‡]

[2]Department of Mechanical and Aerospace Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA, 92093, United States.

[3]George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, 801 Ferst Dr, Atlanta, GA, 30332, United States.

[4]Present address: Center for Control, Dynamical Systems, and Computation, University of California at Santa Barbara, Harold Frank Hall, Room 5119, Santa Barbara, CA, 93106, United States.

[5]Present address: Engineering Sciences Center, Sandia National Laboratories, P.O. Box 5800, MS-1322, Albuquerque, NM, 87185-1322, United States[§]

[‡]This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

[§]This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access

[6]Computational Sciences and Engineering Division, Oak Ridge National Laboratory, P.O. Box 2008, MS-6085, Oak Ridge, TN, 37831-6085, United States.

*Corresponding author(s). E-mail(s): selesonpd@ornl.gov;
Contributing authors: mpasetto@ucsd.edu; yohanjohn@ucsb.edu; jtrages@sandia.gov; reevest@ornl.gov;
†These authors contributed equally to the code development.

Abstract

PDMATLAB2D is a meshfree peridynamics implementation in MATLAB suitable for simulation of two-dimensional fracture problems. The purpose of this code is twofold. First, it provides an entry-level peridynamics computational tool for educational and training purposes. Second, it serves as an accessible and easily modifiable computational tool for peridynamics researchers who would like to adapt the code for a multitude of peridynamics simulation scenarios. The current version of the code implements a bond-based brittle elastic peridynamic model and a critical stretch criterion for bond breaking. However, the code is designed to be extendable for other peridynamic models and computational features. In this paper, we provide an overview of the code structure and functions with illustrative examples. Due to the integrated computation and postprocessing MATLAB capabilities, PDMATLAB2D can serve as an effective testbed for testing new constitutive models and advanced numerical features for peridynamics computations.

Keywords: Peridynamics, Meshfree, Two-dimensional, Fracture

1 Introduction

Peridynamics has become the method of choice for fracture computations for many researchers worldwide. Beginning with its first publication by Silling in 2000 [1], the peridynamics community has grown to encompass hundreds of researchers [2]. With such an increase in the number of peridynamics researchers, the demand for accessible peridynamics computational tools has significantly increased in recent years.

In the earliest years, very few peridynamics codes were publicly available. Among the first such codes was PDLAMMPS (Peridynamics-in-LAMMPS) [3, 4], a peridynamics implementation within the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [5, 6], initially developed in 2008.

to these results of federally sponsored research in accordance with the DOE Public Access Plan (https://www.energy.gov/downloads/doe-public-access-plan).

This was followed by the development of Peridigm [7, 8], made open source in 2011 and built on top of the Trilinos project [9]. Both of these codes are written in C++ and still widely used in the peridynamics community. Since then, many groups have developed research peridynamics codes. Some efforts focus on new algorithms, including fast convolution-based methods [10–12], implementation via the finite element method [13], and integration with the discrete element method [14, 15], but a significant amount of work has also focused on computational performance. Various peridynamics codes have been written in recent years for specific new hardware [16], often NVIDIA GPUs [17, 18], and still others now target many high-performance computing (HPC) systems by leveraging HPC software libraries including OpenCL [19–21], HPX [22, 23], and Kokkos [24, 25]. However, few of these codes focus on an entry-level, easily modifiable tool that can suit the wide variety of needs across disciplines for both research and education. While the principles of meshfree peridynamics computations as laid out in [26] are simple, the intricacies of realistic peridynamics fracture computations, particularly in high dimensions, make full deployment of peridynamics codes non-trivial.

The meshfree method proposed by Silling and Askari in [26] is probably the most commonly used discretization method for peridynamics engineering applications. This meshfree method provides a powerful tool for fracture computations due to its relatively simple implementation and flexible particle-based computation, and it is a discretization method used in PDMATLAB2D. Nevertheless, peridynamics governing equations can be discretized with different discretization methods [27] (some other often used discretization methods include finite elements [28, 29], reproducing kernel [30], and various quadrature rules, e.g., high-order nodal-based quadratures [31, 32]).

In this paper, we describe PDMATLAB2D, a meshfree peridynamics implementation in MATLAB intended for simulation of two-dimensional fracture problems. The current version of the code implements a bond-based brittle elastic peridynamic constitutive model given by a generalization of the prototype microelastic brittle model and bond breaking based on a critical stretch criterion [26] and is available online: https://github.com/ORNL/PDMATLAB2D. This paper discusses version 1.0 of PDMATLAB2D [33]. One of the limitations of peridynamics computations is their computational cost, as compared to more traditional computational mechanics codes [34]. An important feature of PDMATLAB2D which justifies being written in MATLAB, as opposed to lower-level programming languages, is the transparency of the algorithms. Consequently, while many optimizations are possible, we have generally chosen clarity over computational speedup. An additional feature of PDMATLAB2D is that, being based on MATLAB, it provides a convenient single environment for preprocessing, running, and postprocessing peridynamics simulations.

The aim of this paper is to present a comprehensive yet relatively concise description of how peridynamics computations are handled in PDMATLAB2D. This is achieved by first presenting the overall code structure (see Figure 2),

followed by an explanation of the functions of the code with supporting illustrations. For most functions, we provide demonstrations, either in the form of plots or tables, which serve as confirmations of the correctness of their computations; those plots further exemplify different potential adaptations of many of the functions for a variety of purposes. We omit the use of pseudo-codes for most functions, since the MATLAB functions themselves are carefully commented, while only including pseudo-codes in a few instances which we consider necessary for clarity. While the code only incorporates a subset of possible peridynamics computational features, we made an effort to enable extensions, such as implementation of state-based peridynamic models [35].

The outline of this paper is as follows. In Section 2, we describe the bond-based peridynamic theory, a bond-based peridynamic model for brittle elastic fracture, and the meshfree discretization used in PDMATLAB2D. In Section 3, which represents the core of this paper, we describe the PDMATLAB2D code structure and functions. In Section 4, we present numerical examples, demonstrating the applicability of the code on two-dimensional elastic wave propagation and crack branching problems. In Section 5, we provide guidance for some relatively easy code extensions. Finally, in Section 6, we summarize the content of this paper and provide an outlook. A series of appendices are also included which contain relevant complementary materials, including useful analytical derivations, numerical confirmations, and input decks.

2 Bond-based peridynamics

The peridynamic (PD) theory of solid mechanics is a nonlocal reformulation of classical continuum mechanics initially proposed under the *bond-based* form in [1] and generalized to the state-based form in [35]. Here, we concern ourselves only with bond-based PD models. For a review of the PD theory, the reader is referred to [34, 36, 37].

Let us consider a three-dimensional space represented by a Cartesian coordinate system with axes denoted by x, y, and z. To discuss a two-dimensional PD formulation, following [38], we consider a plane stress or plane strain structure of thickness h, represented by a two-dimensional domain Ω. For practical purposes, Ω is assumed to be a subset of the xy-plane within the three-dimensional space. The PD governing equation for a material point $\mathbf{x} \in \Omega$ at time $t \geqslant 0$ can be expressed as follows:[1]

$$\rho(\mathbf{x})\ddot{\mathbf{u}}(\mathbf{x},t) = h\int_{\mathcal{H}_\mathbf{x}} \mathbf{f}(\mathbf{u}(\mathbf{x}',t)-\mathbf{u}(\mathbf{x},t),\mathbf{x}',\mathbf{x},t)d\mathbf{x}' + \mathbf{b}(\mathbf{x},t), \tag{1}$$

where ρ is the mass density, $\ddot{\mathbf{u}}$ is the second derivative in time of the displacement field $\mathbf{u}$, $\mathbf{f}$ is the pairwise force function, and $\mathbf{b}$ is a prescribed body force

[1] We include a separate dependence on $\mathbf{x}$ and $\mathbf{x}'$ as well as an explicit time dependence in the pairwise force function, which is necessary to account for bond breaking.

density field. The spatial integral in (1) is over the neighborhood of $\mathbf{x}$:

$$\mathcal{H}_{\mathbf{x}} := \{\mathbf{x}' \in \Omega : \|\mathbf{x}' - \mathbf{x}\| \leqslant \delta\}, \tag{2}$$

where $\delta > 0$ is the PD horizon and $\|\cdot\|$ represents the Euclidean norm; for points in the bulk of a body, i.e., points farther than δ from the boundary of the body, the neighborhood represents a disk in two dimensions. It is customary to use the notation $\boldsymbol{\xi} := \mathbf{x}' - \mathbf{x}$ and $\boldsymbol{\eta} := \mathbf{u}(\mathbf{x}', t) - \mathbf{u}(\mathbf{x}, t)$, which refer to the relative position in the reference configuration and relative displacement, respectively, of the material points $\mathbf{x}'$ and $\mathbf{x}$; it is also common to refer to $\boldsymbol{\xi}$ as a PD bond. Note that $\boldsymbol{\xi} + \boldsymbol{\eta}$ represents the relative position of $\mathbf{x}'$ and $\mathbf{x}$ in the current configuration. The pairwise force function provides the response of a given bond under deformation in units of force per volume squared. In bond-based PD models, the response of a given bond is independent of the deformation of other bonds.

There are certain conditions imposed on $\mathbf{f}$ to satisfy balance principles [1]. The *linear admissibility condition*,

$$\mathbf{f}(-\boldsymbol{\eta}, \mathbf{x}, \mathbf{x}', t) = -\mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t), \qquad \forall \boldsymbol{\eta}, \mathbf{x}', \mathbf{x} \in \mathbb{R}^3, \quad t \geqslant 0, \tag{3}$$

guarantees conservation of linear momentum. The *angular admissibility condition*,

$$(\boldsymbol{\xi} + \boldsymbol{\eta}) \times \mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = \mathbf{0}, \qquad \forall \boldsymbol{\eta}, \mathbf{x}', \mathbf{x} \in \mathbb{R}^3, \quad t \geqslant 0, \tag{4}$$

guarantees conservation of angular momentum. Elastic materials are represented in peridynamics by microelastic models, which are those that derive from a pairwise potential function (with units of energy per volume squared), w [1]:

$$\mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = \frac{\partial w}{\partial \boldsymbol{\eta}}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t), \qquad \forall \boldsymbol{\eta}, \mathbf{x}', \mathbf{x} \in \mathbb{R}^3, \quad t \geqslant 0. \tag{5}$$

By (3)–(5), a microelastic bond-based PD model is of the following form (see Appendix A):

$$\mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = f(\|\boldsymbol{\xi} + \boldsymbol{\eta}\|, \mathbf{x}', \mathbf{x}, t)\frac{\boldsymbol{\xi} + \boldsymbol{\eta}}{\|\boldsymbol{\xi} + \boldsymbol{\eta}\|}, \tag{6}$$

where f is scalar-valued. The macroelastic energy density of a material point $\mathbf{x} \in \Omega$ at time $t \geqslant 0$ can be express as follows:

$$W(\mathbf{x}, t) = \frac{1}{2}h \int_{\mathcal{H}_{\mathbf{x}}} w(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) d\mathbf{x}'. \tag{7}$$

2.1 The generalized PMB model

The *prototype microelastic brittle* (PMB) PD model was proposed in [26] as a simple brittle elastic PD model. A generalized PMB (GPMB) model was presented in [39, 40]. The generalization is based on incorporation of an influence

function (see Section 2.1.1), which allows control of the bond force and energy based on the bond length.[2] The pairwise force function corresponding to the GPMB model is given by (recall (6))

$$f(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\mathbf{x}',\mathbf{x},t)=\mu(\mathbf{x}',\mathbf{x},t)\,c\,\omega(\|\boldsymbol{\xi}\|)\,s(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|), \tag{8}$$

where ω is an influence function, c is the micromodulus constant, s is the stretch defined as

$$s(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|):=\frac{\|\boldsymbol{\xi}+\boldsymbol{\eta}\|-\|\boldsymbol{\xi}\|}{\|\boldsymbol{\xi}\|}, \tag{9}$$

which represents the relative change in bond length, and μ is a history-dependent Boolean function used for bond breaking. In this paper, we employ a *critical stretch criterion* [26] for bond breaking, where $\mu(\mathbf{x}',\mathbf{x},t)$ takes a value of 1 if $s(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|)<s_0$ for all times $\hat{t}\in[0,t]$ with s_0 a critical stretch and 0 otherwise. With this bond-breaking model, bonds break irreversibly when they exceed a critical stretch, i.e., once a bond is broken, it remains broken for the remainder of a simulation.[3] The corresponding pairwise potential function is

$$w(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\mathbf{x}',\mathbf{x},t)=\frac{1}{2}\mu(\mathbf{x}',\mathbf{x},t)\,c\,\omega(\|\boldsymbol{\xi}\|)\,s^2(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|)\|\boldsymbol{\xi}\|. \tag{10}$$

The micromodulus constant, c, and the critical stretch, s_0, depend on the choice of influence function. The corresponding expressions as functions of the influence function are given in Appendix C, and the specific expressions for the influence functions given in the next section are presented in Section 3.4. The micromodulus constant depends on the Young's modulus E, δ, and h, whereas the critical stretch depends on E, δ, and the fracture energy G_0 (see Tables 2 and 3, respectively).

The Boolean-valued bond-breaking function, μ, is also useful for postprocessing purposes. In particular, we can compute a damage field as [26]

$$\varphi(\mathbf{x},t):=1-\frac{\displaystyle\int_{\mathcal{H}_{\mathbf{x}}}\mu(\mathbf{x}',\mathbf{x},t)d\mathbf{x}'}{\displaystyle\int_{\mathcal{H}_{\mathbf{x}}}d\mathbf{x}'}. \tag{11}$$

Note that $\varphi\in[0,1]$, where 0 represents an undamaged state and 1 indicates a fully damaged state.

[2] While influence functions can be general functions of PD bonds, here we employ spherical influence functions, which only depend on the bond length [35].

[3] Another criteria for bond breaking based on a critical energy density was proposed in [41].

2.1.1 Influence functions

Influence functions are an integral component of PD constitutive models. They have been introduced in [35] and later studied in [39]. Normally, influence functions possess finite support, which often coincides with a PD neighborhood region. However, influence functions can have an independent support from the PD neighborhood, such as support over the entire space. For isotropic materials, influence functions depend only on the bond length, i.e., $\omega = \omega(\|\boldsymbol{\xi}\|)$. Influence functions can be used to modulate the strength of a bond, so that longer bonds may have a weaker strength compared to shorter bonds. Here, we consider influence functions characterized by polynomials of different orders. Specifically, we consider the following influence functions [42]:

$$
\begin{aligned}
\omega_0(r) &:= 1, \\
\omega_{0.5}(r) &:= \begin{cases} 1, & r \leqslant \delta, \\ 0, & \text{else}, \end{cases} \\
\omega_1(r) &:= \begin{cases} 1 - \frac{r}{\delta}, & r \leqslant \delta, \\ 0, & \text{else}, \end{cases} \\
\omega_3(r) &:= \begin{cases} 1 - 3\left(\frac{r}{\delta}\right)^2 + 2\left(\frac{r}{\delta}\right)^3, & r \leqslant \delta, \\ 0, & \text{else}, \end{cases} \\
\omega_5(r) &:= \begin{cases} 1 - 10\left(\frac{r}{\delta}\right)^3 + 15\left(\frac{r}{\delta}\right)^4 - 6\left(\frac{r}{\delta}\right)^5, & r \leqslant \delta, \\ 0, & \text{else}, \end{cases} \\
\omega_7(r) &:= \begin{cases} 1 - 35\left(\frac{r}{\delta}\right)^4 + 84\left(\frac{r}{\delta}\right)^5 - 70\left(\frac{r}{\delta}\right)^6 + 20\left(\frac{r}{\delta}\right)^7, & r \leqslant \delta, \\ 0, & \text{else}, \end{cases}
\end{aligned}
$$

where ω_0 is constant, $\omega_{0.5}$ is piecewise constant, ω_1 is piecewise linear, ω_3 is piecewise cubic, ω_5 is piecewise quintic, and ω_7 is piecewise septic. Note that, except for the constant and piecewise constant ones, these influence functions approach zero when the bond length approaches the PD horizon value. For numerical purposes, this property of influence functions has been leveraged to improve the convergence of PD meshfree discretizations [42–44] to reduce the need of using partial-area or partial-volume corrections; see Section 2.2 for a discussion of partial areas and partial volumes.

2.2 Meshfree PD discretization

A meshfree discretization method for PD governing equations was proposed by Silling and Askari in [26]. In this method, a body Ω is discretized with a set of N computational nodes $\{\mathbf{x}_i\}_{i=1,\dots,N}$, and each node is assigned a corresponding cell or material volume. A uniform grid of nodes is often employed, such as a cubic grid in three dimensions (3D) or a square grid in two dimensions (2D), although sometimes a nonuniform grid is required. In this work, we consider both cases. The PD governing equation is then evaluated at each node via a collocation approach, and the integral over the neighborhood of

the node is discretized as a weighted summation, using neighboring nodes as quadrature points and the corresponding cell volumes (in 3D) or areas (in 2D) as quadrature weights.

In the case of (1), the meshfree discretization produces the following equation for node i at time t^n:

$$\rho_i \ddot{\mathbf{u}}_i^n = h \sum_{j \in \mathcal{F}_i} \mathbf{f}(\mathbf{u}_j^n - \mathbf{u}_i^n, \mathbf{x}_j, \mathbf{x}_i, t^n) A_j + \mathbf{b}_i^n, \tag{12}$$

where $\rho_i := \rho(\mathbf{x}_i)$, $\mathbf{u}_i^n := \mathbf{u}(\mathbf{x}_i, t^n)$, A_j is the area of the cell j, $\mathbf{b}_i^n := \mathbf{b}(\mathbf{x}_i, t^n)$, and $\mathcal{F}_i$ is the family of node i:

$$\mathcal{F}_i := \{j \neq i : \|\mathbf{x}_j - \mathbf{x}_i\| \leqslant \delta\}. \tag{13}$$

To solve peridynamics problems, the discrete governing equation (12) is augmented with initial and boundary conditions, and the time integration is performed via a time-integration scheme (see Section 3.6).

A general quadrature for the numerical computation of the integral over the neighborhood in (1) results in the following equation for node i at time t^n:

$$\rho_i \ddot{\mathbf{u}}_i^n = h \sum_{j \in \hat{\mathcal{F}}_i} \mathbf{f}(\hat{\mathbf{u}}_{j(i)}^n - \mathbf{u}_i^n, \hat{\mathbf{x}}_{j(i)}, \mathbf{x}_i, t^n) \hat{A}_{j(i)} + \mathbf{b}_i^n, \tag{14}$$

where $\hat{\mathcal{F}}_i$ is a set of indices corresponding to the quadrature points, $\hat{\mathbf{x}}_{j(i)}$ is the jth quadrature point, and $\hat{A}_{j(i)}$ is the corresponding jth quadrature weight associated to node i; similar to the above, $\hat{\mathbf{u}}_{j(i)}^n := \mathbf{u}(\hat{\mathbf{x}}_{j(i)}, t^n)$. One of the challenges in using general quadratures is that they may require interpolation procedures, since information for the displacement field at arbitrary points is not readily available. Consequently, nodal-based quadratures that rely on available nodal displacements can be advantageous. To attain such quadratures, a piecewise constant assumption for the displacement field (constant within each nodal cell) can be employed, so that $\hat{\mathbf{u}}_{j(i)}^n = \mathbf{u}_j^n$.

In [42], an algorithm using the areas of the intersections between neighbor cells and the neighborhood of a given source node as quadrature weights (while still using nodes as quadrature points as in (12)) was proposed for the case of uniform grids in 2D, as a means to improve the convergence of PD calculations relative to (12). To account for every cell-neighborhood intersection, this algorithm extends the family of a source node to include all nodes whose cells intersect the neighborhood of the source node[4] (see, e.g., Figure 6b). In such algorithm, $\hat{A}_{j(i)}$ represents the area of the intersection between the cell j and the neighborhood of node i. When the cell j only partially overlaps the neighborhood of node i, the resulting overlapping area is referred to as a *partial area* (PA). The proposed algorithm in [42] employs analytical calculations

[4]Note that the extension of the family of a source node only contributes to the numerical integration for influence functions whose support is nonzero for the added family nodes.

(AC) to compute partial areas exactly. Consequently, the algorithm has been referred to as the PA-AC algorithm. In [42], an improvement to the PA-AC algorithm in terms of convergence of PD calculations was also proposed by using the centroids of the intersections between neighbor cells and the neighborhood of a given source node as quadrature points instead of nodes (see, e.g., Figure 6c). In this case, $\hat{\mathbf{x}}_{j(i)}$ is taken as the centroid of the intersection between the cell j and the neighborhood of node i. The algorithm that computes analytically both the partial areas and the corresponding centroids in [42] is named IPA-AC, where the "I" in the name of the algorithm stands for "improved." In this work, we implement the PA-AC and IPA-AC algorithms in addition to the original meshfree discretization method from [26].

Other algorithms have been proposed in the literature for the computation of partial areas or partial volumes. In 3D, the analogue to a partial area is a partial volume, which refers to the volume of the intersection between a neighbor cell and the neighborhood of a source node. Approaches for numerical approximation of partial areas or partial volumes via the introduction of a linear area/volume scaling or reduction factor appear in [4, 45, 46]. An extension to the PA-AC algorithm, applicable to nonuniform grids in 2D, was presented in [47]. Adaptive schemes to compute partial volumes were proposed based on recursive subdivision and sampling in [43] and trapezoidal integration with error control in [46]. Finally, an analytical approach for the computation of partial areas (a variant of the PA-AC algorithm) and a quasi-analytical approach for the computation of partial volumes were presented in [48].

3 PDMATLAB2D code structure and functions

In this section, we provide an overview of PDMATLAB2D. We begin with a general description of the code structure, followed by a discussion of each of the code functions in Sections 3.1–3.7. The folders and file structure are illustrated in Figure 1.

The code is run by executing the function `PDMATLAB2D`. This can be done (from the top-level directory) by typing in the MATLAB command window:

```
PDMATLAB2D(InputDeck)
```

where `InputDeck` is an input deck located inside the folder `InputFiles`. For instance, the numerical examples for wave propagation in Section 4.1 and crack branching in Section 4.2 can be run with the commands "`PDMATLAB2D('WavePropagation')`" and "`PDMATLAB2D('CrackBranching')`," respectively, where both input decks are located in the `InputFiles` folder (see Figure 1). `PDMATLAB2D` reads the inputs from the input deck and runs the `Main` script, which calls functions located in the `Source` folder (see Figure 2). If outputs (in the form of saved plots) are requested by the user in the input deck, these are saved within the `Outputs` folder inside a subfolder named to match the input deck name.

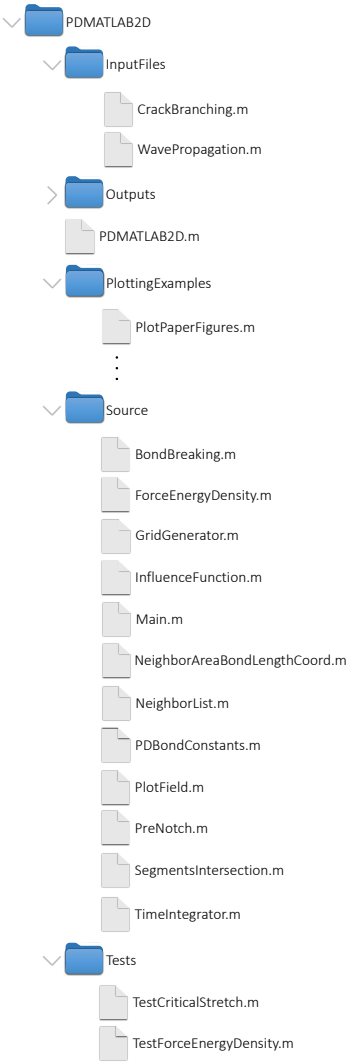


Fig. 1: PDMATLAB2D folders and file structure.

The `Tests` folder contains functions to test and verify the `ForceEnergyDensity` function (see Appendix F) and the critical stretch calculations (see Appendix G). In addition, the `PlottingExamples` folder contains functions to create figures presented from Section 3.1 to Section 3.6 and in Appendix G. For brevity, we do not display all the functions inside the `PlottingExamples` folder in Figure 1. Instead, we only show the function `PlotPaperFigures`, which was created for convenience to produce each of those figures. For instance, to plot Figure 3(a), one simply types the command "`PlotPaperFigures('3(a)')`" within the `PlottingExamples` directory.

As shown in Figure 2, two options are enabled for generation of the grid and neighbor list. In Option I, the user generates the grid and neighbor list outside `Main` and saves those within a file, e.g., using the command

```
save(filename,'xx','yy','u_NA','IF_NA','V_NA','r_hat_NA','x_hat_NA',
    'y_hat_NA')
```

where `filename` is the file name, `'xx'` and `'yy'` are the x- and y-coordinates, respectively, of all the nodes in the grid, and `'u_NA'`, `'IF_NA'`, `'V_NA'`, `'r_hat_NA'`, `'x_hat_NA'`, and `'y_hat_NA'` are neighbor arrays containing the neighbor numbers, influence function values of neighbor bonds, neighbor areas, reference lengths of neighbor bonds, and x- and y-coordinates of quadrature points, respectively (see Section 3.2 for related discussions). Then, the grid is loaded in `Main` by providing the file name in the variable `GridFile` in the input deck. PDMATLAB2D functions can be used to generate grids and neighbor lists outside `Main` (see discussion in Section 3.1.1). In Option II, the user provides inputs for the `GridGenerator` and `NeighborList` functions (see Table 1); in this case, a (regular or irregular) rectangular grid is generated (see discussion in Section 3.1) along with a neighbor list.

The rest of the code structure is described in Figure 2. The code has two major components (indicated by green, dashed-line rectangles): read inputs from `InputDeck` and run `Main`. The `Main` script has several sections, each one represented by a blue, solid-line rectangle. PDMATLAB2D functions are in blue, whereas functions defined by the user in the input deck are in orange. In some cases, PDMATLAB2D functions call other functions; these and relevant operations are indicated within magenta, dashed-dotted line rectangles. In Table 1, we summarize the inputs of an input deck; see Appendices H.1 and H.2 for examples of input decks.

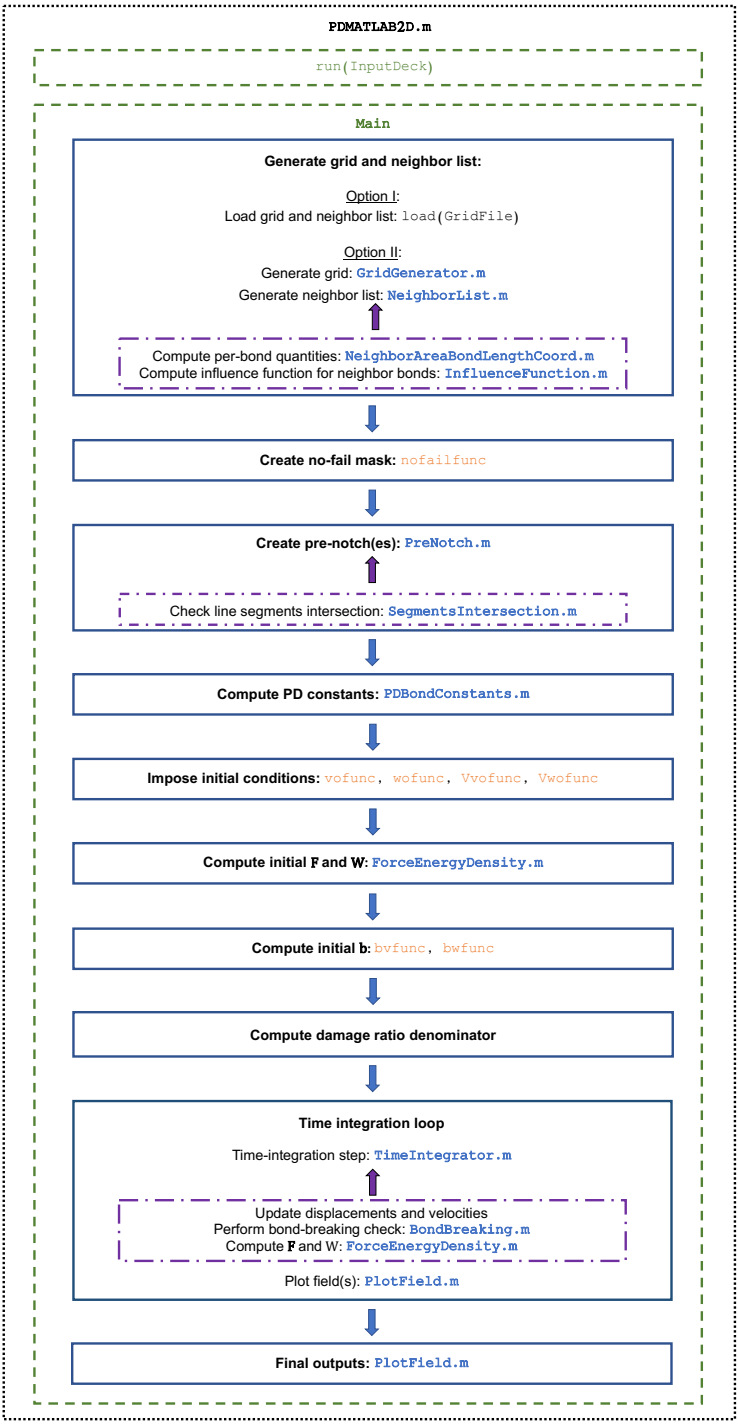


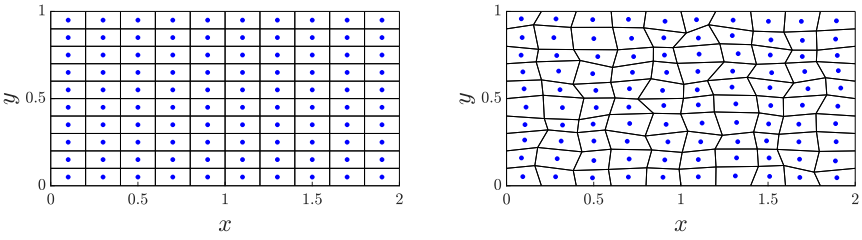
Fig. 2: PDMATLAB2D code structure.

Domain geometry and discretization
Option I
`GridFile`: File name of external grid to load
Option II
`Xo`: Left boundary of the domain `Xn`: Right boundary of the domain `Yo`: Lower boundary of the domain `Yn`: Upper boundary of the domain `Nx`: Number of nodes in the x-direction `Ny`: Number of nodes in the y-direction `PG`: Grid perturbation coefficient
Time discretization
`Ti`: Initial time `Tf`: Final time `dt`: Time step `TimeScheme`: Time-integration scheme ('VVerlet')
PD model
`model`: Constitutive model ('GPMB') `PlanarModel`: Plane elasticity model ('PlaneStrain' or 'PlaneStress') `del`: horizon `omega`: Influence function order indicator (0, 0.5, 1, 3, 5, or 7) `flag_BB`: Flag for bond breaking (0 or 1)
Classical material properties
`rho`: Mass density `E`: Young's modulus `Go`: Fracture energy
Meshfree discretization
`AlgName`: Algorithm for computation of neighbor areas ('FA', 'PA-AC', or 'IPA-AC') (for Option II above)
Problem settings
Body force density
`bvfunc`: x-component of body force density function `bwfunc`: y-component of body force density function
Initial conditions
`vofunc`: x-component of initial displacement function `wofunc`: y-component of initial displacement function `Vvofunc`: x-component of initial velocity function `Vwofunc`: y-component of initial velocity function
No-fail zone (optional)
`nofailfunc`: Function defining a no-fail zone
Prenotch(es) (optional)
`PreNotchCoordinates`: Array with coordinates of pre-notch(es) endpoints (one pre-notch per row)
Postprocessing
`flag_DynamicPlotting`: Flag for plotting during time integration (0 or 1) `DynamicPlotFrequency`: Frequency of plotting during time integration (for `flag_DynamicPlotting` = 1) `TimeStepDisplayFrequency`: Frequency of time-integration step display `flag_FinalPlots`: Flag for plotting at final time `PlotSettings`: Plot settings (array with the following entries: field name (string), field variable (string), colorbar title (string), point size, colormap limits (1×2 array), colormap (string), axes limits (1×4 array), configuration ('Reference' or 'Current')) `flag_DamagedPrenotches`: Flag to visualize pre-notch(es) as damaged (0 or 1) (for `flag_BB` = 1)

Table 1: Description of inputs for an input deck.

3.1 GridGenerator

The function `GridGenerator` generates a grid of nodes over a rectangular domain. The grid contains information about the nodal positions as well as the vertices and area of the cell surrounding each node. The function allows generation of either a regular grid or an irregular grid, where the irregular grid is obtained by perturbing the cell vertices inside the domain (while keeping the vertices on the domain boundary unperturbed) and then recomputing the nodal positions and cell areas associated with the resulting cells; the grid perturbation in each dimension is less than a quarter of the corresponding grid spacing to guarantee that the resulting cells remain convex (see Appendix B). The perturbed cell areas are obtained by computing the areas of the resulting quadrilaterals, and the nodal positions are updated as the centroids of those quadrilaterals. The tessellation generated by the set of cells is such that it fully covers the domain. Examples of regular and irregular grids generated by the `GridGenerator` function are presented in Figure 3.



(a) Regular grid. (b) Irregular grid.

Fig. 3: Grids generated by the `GridGenerator` function. The figures are produced with the `PlotGrid` function.

Remark 1 The grid generated by the `GridGenerator` function is based on input parameters defining the number of cells per dimension. For a regular grid, this results in all cells having the same area and the set of cells fully covering the domain. On the other side, the discretization may not result in uniform (square) cells, which are often desired in peridynamics discretizations (such as for partial-area calculations, see Section 2.2). To ensure a uniform discretization based on square cells, one could instead use as inputs the grid spacings, Δx and Δy for the x- and y-directions, respectively. However, this alternative discretization approach may result in the physical domain not exactly coinciding with the domain represented by the discretization, and boundary corrections would be required. Therefore, this discretization approach is not used by the `GridGenerator` function.

Remark 2 While the `GridGenerator` function only produces (possibly perturbed) rectangular grids, PDMATLAB2D is designed to handle general grids, such as the one

presented in Figure 5f. Moreover, by combining various functions of PDMATLAB2D, one can generate complex shapes, as demonstrated in Section 3.1.1.

3.1.1 Generation of complex shapes

This section demonstrates how a combined use of the functions `GridGenerator`, `NeighborList` (see Section 3.2), and `Prenotch` (see Section 3.3) can generate far more complex shapes than the simple rectangular geometries generated by the `GridGenerator` function. To provide a variety of complex shape flavors, we present three examples:

1. L-shape domain,
2. Circular domain,
3. Square domain with a circular hole at its center.

Below, we describe the specifics of how each of these shapes are generated. These examples are illustrated in Figure 4. The FA algorithm is employed to generate the neighbor lists.

L-shape domain

The L-shape domain illustrated in Figures 4a and 4d is generated via the following steps:

1. We generate two rectangular subdomains: $\Omega_1 = (0, 2) \times (0, 1)$ and $\Omega_2 = (0, 1) \times (1, 2)$ with a regular grid of $\Delta x = 0.2$ and $\Delta y = 0.1$ by calling the `GridGenerator` function twice.
2. We combine the resulting grids into a single grid.
3. We generate the neighbor list by calling the `NeighborList` function with a choice of $\delta = 2\Delta x$.
4. We break all the bonds crossing the domain boundary by defining a prenotch vertical segment with endpoints $(1, 1)$ and $(1, 2)$ and calling the `PreNotch` function.

Circular domain

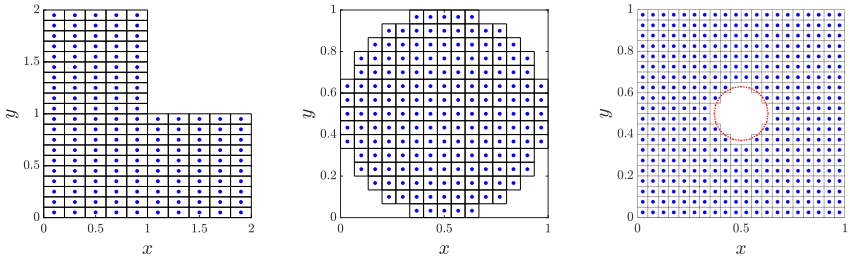
The circular domain illustrated in Figures 4b and 4e is generated via the following steps:

1. We generate a square domain: $\Omega_1 = (0, 1) \times (0, 1)$ with a uniform grid of grid spacing $\Delta x = \Delta y = 1/15$ by calling the `GridGenerator` function.
2. We remove all nodes outside a circle of radius $R = 0.5$ centered at $(0.5, 0.5)$.
3. We remove all cell vertices that do not belong to cells corresponding to nodes inside the circle.
4. We generate the neighbor list by calling the `NeighborList` function with a choice of $\delta = 2\Delta x$.

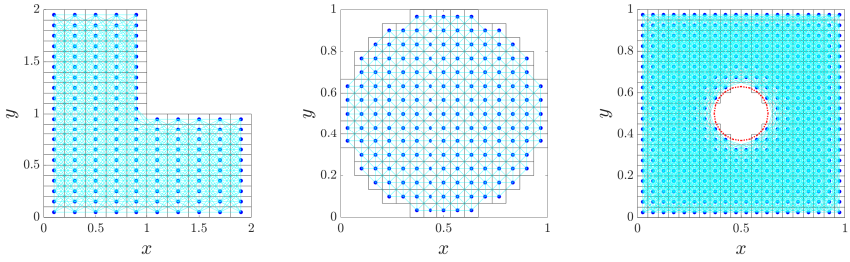
Square domain with a circular hole at its center

The square domain with a circular hole at its center illustrated in Figures 4c and 4f is generated via the following steps:

1. We generate a square domain: $\Omega_1 = (0, 1) \times (0, 1)$ with a uniform grid of grid spacing $\Delta x = \Delta y = 0.05$ by calling the `GridGenerator` function.
2. We remove all nodes inside a circle of radius $R = 0.13$ centered at $(0.5, 0.5)$.
3. We remove all cell vertices that do not belong to cells corresponding to nodes outside the circle.
4. We generate the neighbor list by calling the `NeighborList` function with a choice of $\delta = 3.5\Delta x$.
5. We approximate the hole as an icosagon (a twenty-sided polygon) and we consider each side of the icosagon as a "pre-notch". We loop over each of the polygon's sides and break all the bonds intersecting each side by defining a pre-notch with the side's endpoints and calling the `PreNotch` function. In Figures 4c and 4f, the icosagon is illustrated with a red dotted thick line.



(a) L-shape domain. (b) Circular domain. (c) Square domain with circular hole.



(d) L-shape domain with bonds. (e) Circular domain with bonds. (f) Square domain with circular hole with bonds.

Fig. 4: Illustration of combining the `GridGenerator`, `NeighborList`, and `Prenotch` functions to generate complex shapes. The domains are: an L-shape domain (left column), a circular domain (middle column), and a square domain with a circular hole at its center (right column). The top figures show the corresponding grids for each shape, while the bottom figures present in addition all the generated bonds in cyan. The figures are produced with the `PlotComplexShapes` function.

3.2 `NeighborList`

The function `NeighborList` generates a neighbor list and computes per-bond quantities. It employs the function `NeighborAreaBondLengthCoord` (see Section 3.2.1) to compute neighbor areas, reference lengths of neighbor bonds, and coordinates of quadrature points, and it uses the function `InfluenceFunction` (see Section 3.2.2) to compute influence function values of neighbor bonds. The function considers two cases:

1. Uniform grids over rectangular domains.
2. General grids.

Below, we describe how the function loops over cells surrounding a source node to generate the neighbor list and compute per-bond quantities in each case.

Uniform grids over rectangular domains

In this case, the function employs an efficient algorithm to loop over the cells that overlap the neighborhood of a given source node, following [42] (see "Algorithm PA-AC Family Interaction" in that reference), as opposed to looping over all cells in the system for each source node.

General grids

In this case, since information about the grid or domain is not directly available, the function loops over all cells in the system for each source node.

In Figure 5, we demonstrate the function's computations by presenting examples of discretized domains with all bonds drawn. To make the distribution of bonds per computational node more clear, we select a particular source node and emphasize all its bonds as well as the boundary of its neighborhood. The top plots illustrate uniform grids over rectangular domains, differentiating between different algorithms for computing quadrature points: FA algorithm (Figure 5a), PA-AC algorithm (Figure 5b), and IPA-AC algorithm (Figure 5c). The bottom plots, in contrast, show nonuniform grids: a rectangular grid (Figure 5d), a perturbed rectangular grid (Figure 5e), and an irregular grid over a circular domain (Figure 5f).

3.2.1 `NeighborAreaBondLengthCoord`

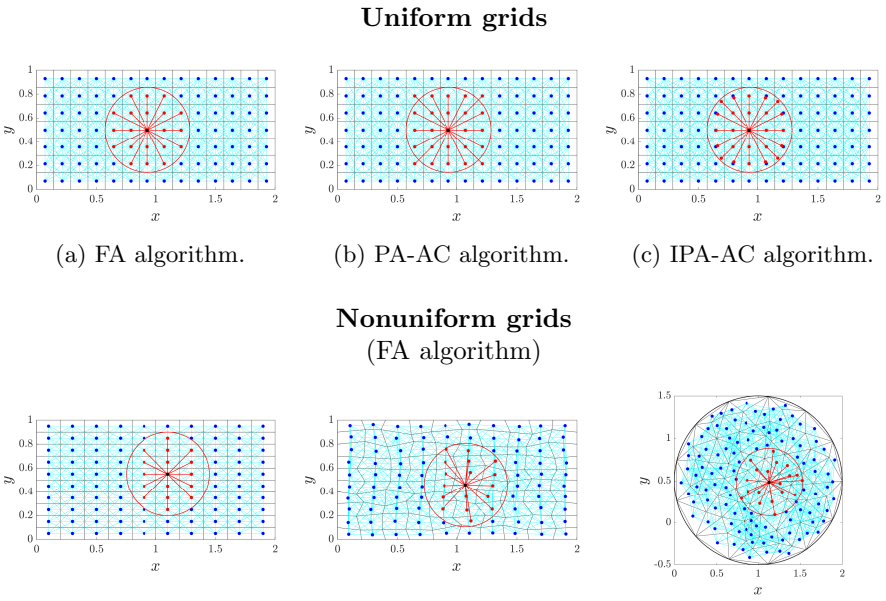
The function `NeighborAreaBondLengthCoord` computes the following quantities associated to the neighboring quadrature point $\hat{\mathbf{x}}_{j(i)}$ in (14):

1. neighbor area (employed as the quadrature weight, $\hat{A}_{j(i)}$, in (14)),
2. bond length relative to a source node i: $\|\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i\|$,
3. coordinates of the quadrature point.

Below, we describe the various computation options enabled by this function.

Computation of the neighbor area

The function allows two options for the neighbor areas:



(a) FA algorithm. (b) PA-AC algorithm. (c) IPA-AC algorithm.

(d) Regular grid. (e) Perturbed regular grid. (f) Irregular grid

Fig. 5: Illustration of discretized domains with all bonds drawn in cyan. Grid nodes are in blue. The neighborhood of a single source node (in black) is emphasized with neighbor bonds and quadrature points as well as the boundary of the neigborhood in red. The top plots illustrate the three algorithms for computing quadrature points (FA, PA-AC, and IPA-AC) for the case of uniform grids. The bottom plots are for nonuniform grids and employ the FA algorithm, and they illustrate three types of grids: regular, perturbed regular, and irregular. The list of bonds per node is computed with the `NeighborList` function. The figures are produced with the `PlotNeighborList` function.

- Full Area (FA): The neighbor area associated with a neighboring quadrature point j is simply taken as the area of the cell j.
- Partial Area (PA): The neighbor area associated with a neighboring quadrature point j is taken as the area (based on analytical calculations) of the intersection between the cell j and the neighborhood of a source node i.

Computation of the coordinates of the quadrature point

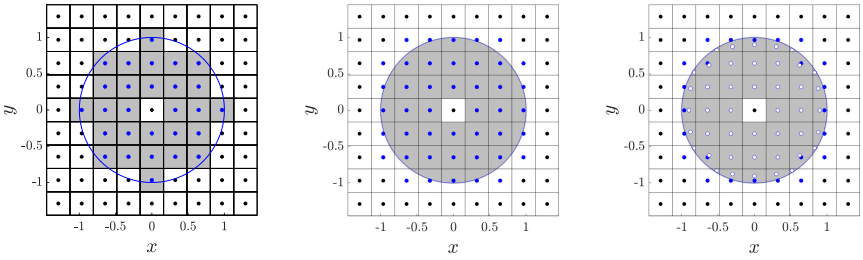
The function allows two options for the coordinates of quadrature points:

- Nodal coordinates: The coordinates of the quadrature point are simply taken as the coordinates of node j.
- Centroid coordinates: The coordinates of the quadrature point are taken as the coordinates of the centroid (based on analytical calculations) of the intersection between the cell j and the neighborhood of a source node i.

The following algorithms, which can be selected in the function, combine the computation of the neighbor area and coordinates of the quadrature point:

- FA algorithm: FA option in combination with nodal coordinates.
- PA-AC algorithm: PA option in combination with nodal coordinates.
- IPA-AC algorithm: PA option in combination with centroid coordinates.

These algorithms are illustrated in Figure 6.



(a) FA algorithm. (b) PA-AC algorithm. (c) IPA-AC algorithm.

Fig. 6: Illustration of the family and neighbor areas of a source node located at the origin for a PD horizon $\delta = 1$ and a uniform grid with grid spacing $\Delta x = \Delta y = \delta/3.1$. Family nodes are represented by blue dots and centroids are represented by white dots (in (c)); for cells fully contained within the neighborhood, centroids coincide with family nodes. The boundary of the neighborhood is represented by a blue circle. Cell areas and coordinates of quadrature points are computed with the `NeighborAreaBondLengthCoord` function. The figures are produced with the `PlotNeighborAreasCoords` function.

3.2.2 `InfluenceFunction`

The function `InfluenceFunction` computes the influence function. It implements the influence functions described in Section 2.1.1. These influence functions are illustrated in Figure 7.

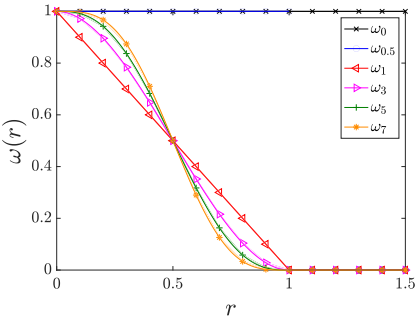
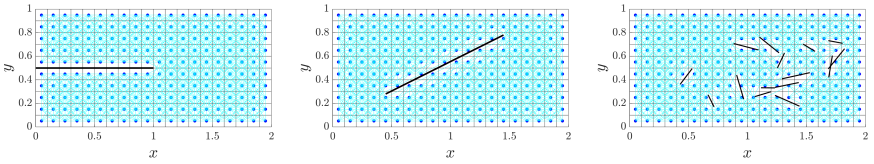


Fig. 7: Influence functions computed with the `InfluenceFunction` function for a choice of $\delta = 1$. The figure is produced with the `PlotInfluenceFunction` function.

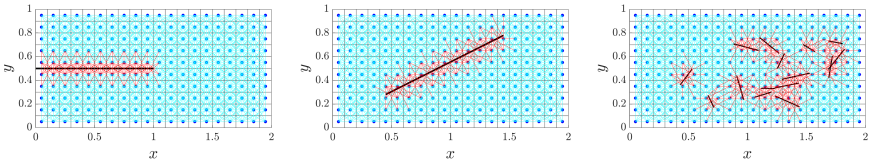
3.3 `PreNotch`

The function `PreNotch` creates a pre-notch described by an arbitrary line segment. A pre-notch is created by defining a line segment representing it, and then breaking all bonds that either cross the line segment or overlap it. For this purpose, the function checks for each intact bond whether it intersects the pre-notch. This is achieved, in practice, by looping over all nodes and all their neighbors and checking whether there is an intersection between the two line segments defined by the bond and the pre-notch, using the function `SegmentsIntersection` (see Section 3.3.1). When such an intersection occurs for a given bond connecting a node i and the quadrature point corresponding to the cell j, the function removes the cell j from the neighbor list of node i and the cell i from the neighbor list of node j by replacing the cell numbers by 0.

To illustrate the creation of pre-notches, Figure 8 presents three examples: a horizontal pre-notch (left column), an inclined pre-notch (middle column), and multiple random pre-notches (right column). The domain is $\Omega = (0, 2) \times (0, 1)$, and it is discretized with a uniform grid of grid spacing $\Delta x = \Delta y = 0.1$. For simplicity, we employ the FA algorithm. In the case of the horizontal pre-notch, the pre-notch is defined by the line segment connecting the points $(0.0, 0.5)$ and $(1.0, 0.5)$. In the case of the inclined pre-notch, the pre-notch is defined by the line segment connecting the points $(0.45, 0.28)$ and $(1.45, 0.78)$. In the case of the multiple random pre-notches, we generate 15 pre-notches of random length l with $l \in (0.1, 0.25)$, randomly located and oriented within the domain, as follows. For each pre-notch, we randomly choose an endpoint farther apart from the domain boundaries than 0.25 to ensure the pre-notch doesn't extend outside the domain. Then, we choose the other endpoint by randomly choosing the line segment length and orientation. The figure shows the domain with the intact bonds in cyan (top plots) as well as both the intact bonds in cyan and broken bonds in red (bottom plots).



(a) Horizontal pre-notch (intact bonds only). (b) Inclined pre-notch (intact bonds only). (c) Random pre-notches (intact bonds only).



(d) Horizontal pre-notch (intact and broken bonds). (e) Inclined pre-notch (intact and broken bonds). (f) Random pre-notches (intact and broken bonds).

Fig. 8: Illustration of creation of pre-notches in a rectangular domain discretized with a uniform grid. Three cases are presented: a horizontal pre-notch (left column), an inclined pre-notch (middle column), and multiple random pre-notches (right column). Grid nodes are in blue, and the line segments representing the pre-notches are plotted with black solid thick lines. Intact bonds are shown in cyan (top and bottom plots) while broken bonds are shown in red (bottom plots). The pre-notches are generated with the `PreNotch` function, which is called repeated times for the case of multiple random pre-notches. The figures are produced with the `PlotPreNotch` function.

3.3.1 `SegmentsIntersection`

The function `SegmentsIntersection` checks whether two line segments intersect, following the derivations in Appendix E. In Algorithm 1, we present a pseudo-algorithm describing the procedure used to detect the intersection of two arbitrary line segments. Note that we employ the notation $(\mathbf{a} \times \mathbf{b})_z$ to denote the z-component of the cross product $\mathbf{a} \times \mathbf{b}$.

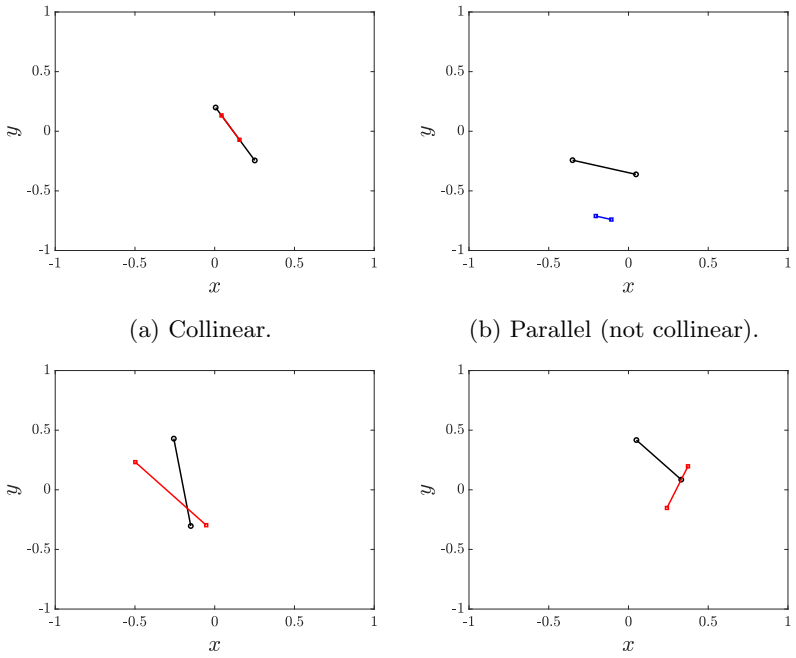
To illustrate the performance of the `SegmentsIntersection` function, we present in Figure 9 four cases of two line segments and their intersection or lack of intersection. The first case presents two collinear line segments (Figure 9a). The second case presents two parallel but not collinear line segments (Figure 9b). The third and four cases present two non-parallel line segments, showing a general case (Figure 9c) as well as a case where the second line segment intersects an endpoint of the first line segment (Figure 9d). An intersection case is highlighted by plotting one of the line segments in red while plotting it in blue otherwise.

Algorithm 1 Intersection of two line segments

Given two line segments defined by $\mathbf{x}_{1A}$ & $\mathbf{x}_{2A}$ and $\mathbf{x}_{1B}$ & $\mathbf{x}_{2B}$, respectively
flag_intersection = 0 ▷ initialize intersection flag
Define 1st line: $\mathbf{p} + t\,\mathbf{r}$ with $\mathbf{p} := \mathbf{x}_{1A}$ and $\mathbf{r} := \mathbf{x}_{2A} - \mathbf{x}_{1A}$, $t \in (-\infty, \infty)$
Define 2nd line: $\mathbf{q} + u\,\mathbf{s}$ with $\mathbf{q} := \mathbf{x}_{1B}$ and $\mathbf{s} := \mathbf{x}_{2B} - \mathbf{x}_{1B}$, $u \in (-\infty, \infty)$
{Check if lines are parallel}
if $(\mathbf{r} \times \mathbf{s})_z = 0$ **then**
    {Lines are parallel: check if lines are collinear}
    if $((\mathbf{p} - \mathbf{q}) \times \mathbf{s})_z = 0$ **then**
        {Lines are collinear}
        {Express endpoints $\mathbf{p}$ and $\mathbf{p}+\mathbf{r}$ of 1st segment in terms of 2nd line}
        $u_0 = \frac{(\mathbf{p}-\mathbf{q})\cdot\mathbf{s}}{\mathbf{s}\cdot\mathbf{s}}$
        $u_1 = u_0 + \frac{\mathbf{r}\cdot\mathbf{s}}{\mathbf{s}\cdot\mathbf{s}}$
        {Check if segments overlap}
        if $\min(u_0, u_1) > 1$ or $\max(u_0, u_1) < 0$ **then**
            {Segments do not overlap}
        else
            {Segments overlap}
            flag_intersection = 1
        end if
    else
        {Lines are parallel but not collinear: segments do not overlap}
    end if
else
    {Lines are not parallel: solve for t_0 and u_0}
    $t_0 = \frac{((\mathbf{q}-\mathbf{p})\times\mathbf{s})_z}{(\mathbf{r}\times\mathbf{s})_z}$ ▷ parameter of 1st line
    $u_0 = \frac{((\mathbf{q}-\mathbf{p})\times\mathbf{r})_z}{(\mathbf{r}\times\mathbf{s})_z}$ ▷ parameter of 2nd line
    {Check if segments intersect}
    if $u_0, t_0 \in [0, 1]$ **then**
        {Segments intersect}
        flag_intersection = 1
    else
        {Segments are not parallel but they do not intersect}
    end if
end if

3.4 PDBondConstants

The function `PDBondConstants` computes the micromodulus constant c and the critical stretch s_0 for the GPMB model (see Section 2.1). These two PD bond constants are different for the case of plane strain and plane stress, and they depend on the choice of influence function. The micromodulus constant depends on the Young's modulus E, horizon, δ, and thickness, h, whereas the critical stretch depends on E, δ, and the fracture energy G_0. The derivations



(a) Collinear.

(b) Parallel (not collinear).

(c) Non-parallel.

(d) Non-parallel with endpoint intersection.

Fig. 9: Illustration of different cases of two line segments and their intersection check. We present four cases for the two line segments: collinear, parallel, non-parallel, and non-parallel with an endpoint intersection. The first line segment is plotted in black. The second line segment is plotted in red for the case of an intersection, while it is plotted in blue for the case of no intersection. The intersection check is performed with the function `SegmentsIntersection`. The figures are produced with the `TestSegmentsIntersection` function.

for these PD bond constants are presented in Appendix C, and the resulting expressions are reported for the micromodulus constant in Table 2 and for the critical stretch in Table 3.

	ω_0	ω_1	ω_3	ω_5	ω_7
Plane strain c	$\frac{48E}{5\pi h\delta^3}$	$\frac{192E}{5\pi h\delta^3}$	$\frac{48E}{\pi h\delta^3}$	$\frac{1344E}{25\pi h\delta^3}$	$\frac{288E}{5\pi h\delta^3}$
Plane stress c	$\frac{9E}{\pi h\delta^3}$	$\frac{36E}{\pi h\delta^3}$	$\frac{45E}{\pi h\delta^3}$	$\frac{252E}{5\pi h\delta^3}$	$\frac{54E}{\pi h\delta^3}$

Table 2: Micromodulus constant expressions for plane strain and plain stress for different choices of influence functions.

	ω_0	ω_1	ω_3	ω_5	ω_7
Plane strain s_0	$\sqrt{\frac{5\pi G_0}{12E\delta}}$	$\sqrt{\frac{25\pi G_0}{48E\delta}}$	$\sqrt{\frac{7\pi G_0}{12E\delta}}$	$\sqrt{\frac{5\pi G_0}{8E\delta}}$	$\sqrt{\frac{55\pi G_0}{84E\delta}}$
Plane stress s_0	$\sqrt{\frac{4\pi G_0}{9E\delta}}$	$\sqrt{\frac{5\pi G_0}{9E\delta}}$	$\sqrt{\frac{28\pi G_0}{45E\delta}}$	$\sqrt{\frac{2\pi G_0}{3E\delta}}$	$\sqrt{\frac{44\pi G_0}{63E\delta}}$

Table 3: Critical stretch expressions for plane strain and plain stress for different choices of influence functions.

A numerical confirmation can be performed for the expressions of micromodulus constant and critical stretch. In Appendix F, we report values for the numerical computation of the macroelastic energy density, which serve as confirmation for the expressions of the micromodulus constant. In Appendix G, we present numerical computations for the fracture energy to confirm the expressions of the critical stretch.

3.5 `ForceEnergyDensity`

The function `ForceEnergyDensity` computes the internal force density (the first term on the right-hand side of (1)) and the macroleastic energy density (7) at each grid node based on the GPMB model (see (8) and (10)); the `PDBondConstants` function (see Section 3.4) is used to compute the micromodulus constant c of the GPMB model. Similar to the function `NeighborList`, the `ForceEnergyDensity` function considers two cases:

1. Uniform grids over rectangular domains,
2. General grids.

Below, we describe how the function handles each case.

Uniform grids over rectangular domains

In this case, the function employs the linear admissibility condition (3) to reduce the number of computations, as follows. Given a source node i, when the pairwise force function and pairwise potential function are evaluated with respect to the quadrature point corresponding to the cell j, the linear admissibility condition is used to directly assign the corresponding values for the node j with respect to the quadrature point corresponding to the cell i. In Algorithm 2, we present a pseudo-algorithm describing the general procedure for the computation of the internal force density and macroelastic energy density for all nodes in a system. Note that in Line 10 we only perform computations for $j > i$, leveraging the linear admissibility condition.

Remark 3 The computation of the internal force density and macroelastic energy density requires knowledge of the reference bond lengths and neighbor areas which, in general, can be different for the node i with respect to the cell j and for the node j with respect to the cell i; however, these quantities are the same for the case of uniform grids over rectangular domains, reducing the amount of information required for these computations. While the strategy of using half the bond computations is

possible for the case of general grids, given a source node i, seeking the neighbor area of its cell relative to the node j adds significant overhead to the computations.

General grids

In this case, since the neighbor areas may not be symmetric for all bonds (see Remark 3), the function computes the pairwise force function and pairwise potential function separately for the bond connecting node i to the cell j and the bond connecting node j to the cell i. The computation of the internal force density and macroelastic energy density in this case is identical to Algorithm 2 except for a couple of changes. First, in Line 10, the condition $j > i$ is replaced by $j > 0$. Second, we omit the updates for node j in Line 19 and Line 23. The relevant expressions are highlighted in blue in Algorithm 2.

Remark 4 We note that in Algorithm 2 the history-dependent Boolean function μ used for bond breaking is absent. The reason for this is that, in practice, when a bond breaks, the corresponding neighbor cell number is replaced by 0 in the neighbor list (see Section 3.6.1). Therefore, it is implicitly assumed that $\mu = 1$.

In Appendix F, we present numerical computations for the internal force density and macroelastic energy density to test and verify the `ForceEnergyDensity` function. These computations are carried out with the `TestForceEnergyDensity` function.

3.6 `TimeIntegrator`

The function `TimeIntegrator` performs a time-integration step. Given quantities at time step n, the `TimeIntegrator` function computes quantities at time step $n + 1$. The function uses the standard velocity Verlet time-integration scheme, which can be summarized with the following three equations:

$$\dot{\mathbf{u}}_i^{n+\frac{1}{2}} = \dot{\mathbf{u}}_i^n + \frac{\Delta t}{2}\ddot{\mathbf{u}}_i^n, \tag{15a}$$

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n + \Delta t\, \dot{\mathbf{u}}_i^{n+\frac{1}{2}}, \tag{15b}$$

$$\dot{\mathbf{u}}_i^{n+1} = \dot{\mathbf{u}}_i^{n+\frac{1}{2}} + \frac{\Delta t}{2}\ddot{\mathbf{u}}_i^{n+1}, \tag{15c}$$

where $\mathbf{u}_i^n$, $\dot{\mathbf{u}}_i^n$, and $\ddot{\mathbf{u}}_i^n$ are, respectively, the displacement, velocity, and acceleration of node i (with reference position $\mathbf{x}_i$) at time t^n. The acceleration of node i at time step n can be further computed as (see (14)):

$$\ddot{\mathbf{u}}_i^n = \frac{\mathbf{F}_i^n + \mathbf{b}_i^n}{\rho_i},$$

Algorithm 2 Computation of internal force density and macroleastic energy density for the GPMB model (for uniform grids over rectangular domains)

Find number of nodes N_{nodes} from nodal positions array
Initialize internal force density $\mathbf{F}$ for all nodes
Initialize macroelastic energy density W for all nodes
Find maximum number of neighbors per node $z_{\max}$ from neighbor list
for $i = 1$ to N_{nodes} **do**
    Get node i reference position: $\mathbf{x}_i$
    Get node i displacement: $\mathbf{u}_i$
    for $z = 1$ to $z_{\max}$ **do**
        Get neighbor cell number j from neighbor list
        if $j > i$ **then**
            Get cell j quadrature point: $\hat{\mathbf{x}}_{j(i)}$
            Get node j displacement: $\mathbf{u}_j$
            Get per-bond quantities: $\omega(\|\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i\|)$, $\hat{A}_{j(i)}$, and $\|\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i\|$
            $s = \frac{\|(\mathbf{u}_j - \mathbf{u}_i) + (\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i)\| - \|\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i\|}{\|\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i\|}$ ▷ stretch (see (9))
            $f = c\,\omega(\|\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i\|)s$ ▷ pairwise force magnitude (see (8))
            $\mathbf{f} = f\frac{(\mathbf{u}_j - \mathbf{u}_i) + (\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i)}{\|(\mathbf{u}_j - \mathbf{u}_i) + (\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i)\|}$ ▷ pairwise force (see (6))
            {Update nodal internal force density}
            node i: $\mathbf{F}(i) = \mathbf{F}(i) + \mathbf{f}\hat{A}_{j(i)}$
            node j: $\mathbf{F}(j) = \mathbf{F}(j) - \mathbf{f}\hat{A}_{j(i)}$
            $w = \frac{1}{2}c\,\omega(\|\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i\|)s^2\|\hat{\mathbf{x}}_{j(i)} - \mathbf{x}_i\|$ ▷ pairwise potential (see (10))
            {Update nodal macroelastic energy density}
            node i: $W(i) = W(i) + \frac{1}{2}w\hat{A}_{j(i)}$
            node j: $W(j) = W(j) + \frac{1}{2}w\hat{A}_{j(i)}$
        end if
    end for
end for

where $\mathbf{F}_i^n$ is the internal force density (the first term on the right-hand side of (14)) at node i and time t^n. Note that numerical computation of the internal force density at time step n relies on knowledge about which bonds are intact at that time step. Therefore, before computing the internal force density, the `TimeIntegrator` function checks whether bonds need to be broken based on the status of the deformation of all nodes at time step n and breaks critically stretched bonds (assuming a bond-breaking check is enabled by a simulation) with the `BondBreaking` function (see Section 3.6.1). Moreover, for all nodes i at time t^n, the macroelastic energy density W_i^n is computed (mainly for postprocessing purposes) in addition to $\mathbf{F}_i^n$. Algorithm 3, summarizes the operations for a time-integration step. Numerical examples in Section 4 confirm the performance of the `TimeIntegrator` function.

Algorithm 3 Time-integration step: velocity Verlet with bond breaking

1: Given ρ, $\mathbf{u}^n$, $\dot{\mathbf{u}}^n$, $\mathbf{F}^n$, and $\mathbf{b}^n$ for all nodes at time t^n
2: {Compute velocity at time $t^{n+\frac{1}{2}} := t^n + \frac{\Delta t}{2}$ for all nodes}
3: $\dot{\mathbf{u}}^{n+\frac{1}{2}} = \dot{\mathbf{u}}^n + \frac{\Delta t}{2}(\mathbf{F}^n + \mathbf{b}^n)/\rho$
4: {Compute displacement at time $t^{n+1} := t^n + \Delta t$ for all nodes}
5: $\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t\, \dot{\mathbf{u}}^{n+\frac{1}{2}}$
6: Check bond breaking for all bonds with the `BondBreaking` function
7: Compute $\mathbf{F}^{n+1}$ and W^{n+1} for all nodes with the `ForceEnergyDensity` function
8: Compute $\mathbf{b}^{n+1}$ for all nodes based on provided functions
9: {Compute velocity at time $t^{n+1} := t^n + \Delta t$ for all nodes}
10: $\dot{\mathbf{u}}^{n+1} = \dot{\mathbf{u}}^{n+\frac{1}{2}} + \frac{\Delta t}{2}(\mathbf{F}^{n+1} + \mathbf{b}^{n+1})/\rho$

Remark 5 We note that the velocity Verlet time-integration scheme (15) is equivalent to the second-order centered finite difference approximation of the second time derivative of the displacement field (see, e.g., [49]), which for node i and time t^n is:

$$\frac{\mathbf{u}_i^{n+1} - 2\mathbf{u}_i^n + \mathbf{u}_i^{n-1}}{(\Delta t)^2} = \ddot{\mathbf{u}}_i^n. \tag{16}$$

To show this, take (15c) for $\dot{\mathbf{u}}_i^n$:

$$\dot{\mathbf{u}}_i^n = \dot{\mathbf{u}}_i^{n-\frac{1}{2}} + \frac{\Delta t}{2}\ddot{\mathbf{u}}_i^n$$

and substitute it in (15a) to get:

$$\dot{\mathbf{u}}_i^{n+\frac{1}{2}} = \dot{\mathbf{u}}_i^{n-\frac{1}{2}} + \Delta t \ddot{\mathbf{u}}_i^n.$$

Then, use (15b) to substitute expressions for $\dot{\mathbf{u}}_i^{n+\frac{1}{2}}$ and $\dot{\mathbf{u}}_i^{n-\frac{1}{2}}$:

$$\frac{\mathbf{u}_i^{n+1} - \mathbf{u}_i^n}{\Delta t} = \frac{\mathbf{u}_i^n - \mathbf{u}_i^{n-1}}{\Delta t} + \Delta t \ddot{\mathbf{u}}_i^n,$$

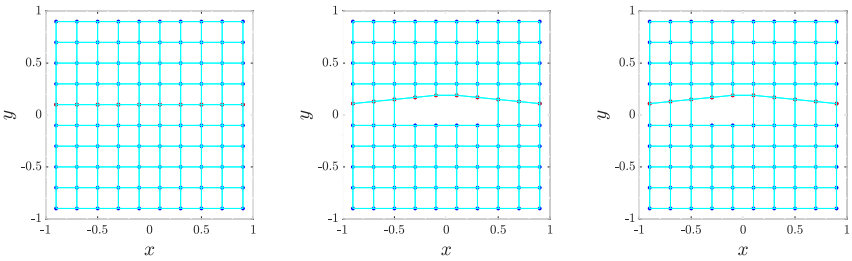
from which (16) is obtained. Note that for (16), to compute the displacement of node i at time step $n+1$, $\mathbf{u}_i^{n+1}$, one requires information about the displacement at two previous time steps, $\mathbf{u}_i^n$ and $\mathbf{u}_i^{n-1}$. In contrast, in (15) one requires instead information about the displacement and velocity at the previous time step only, $\mathbf{u}_i^n$ and $\dot{\mathbf{u}}_i^n$, respectively. However, we observe that these two different data can be related using the above equations. A stability analysis for (16) has been derived in [26, 50] using the case of a one-dimensional bar represented by a linear PD model discretized with the original meshfree discretization method from [26].

3.6.1 `BondBreaking`

The function `BondBreaking` breaks all the bonds with stretch s exceeding a critical stretch, s_0. Since the options for combined grid types and algorithms for neighbor areas result in symmetric quantities for reference and deformed bond

lengths between the node i and the quadrature point associated to the cell j (relative to the node i) and the node j and the quadrature point associated to the cell i (relative to the node j), if the condition for bond breaking is met for either of these cases, both bonds get broken. In practice, the function breaks bonds by replacing the corresponding neighbor cell number in the neighbor list by 0. Following [51], the function enables the definition of a no-fail zone, which contains points for which their bonds remain intact; these points can be grid nodes as well as quadrature points.

To illustrate the bond breaking and the effect of the no-fail zone, in Figure 10 we present an example of a domain discretized with a uniform grid, where a single horizontal line of nodes is deformed and bonds exceeding a critical stretch (except for those connected to a no-fail zone, as in Figure 10c) get broken. The domain is $\Omega = (-1, 1) \times (-1, 1)$, and the grid spacing is chosen as $\Delta x = \Delta y = 0.2$. The line of deformed nodes is selected as the first horizontal line of nodes with positive y coordinates. The deformation for the selected nodes is a linear deformation in the y-direction given by $\mathbf{u}(x, y) = (0, \frac{1}{2}\Delta y(1 - |x|))$, and the critical stretch for bond breaking is chosen as $s_0 = 0.25$. For simplicity, we choose $\delta = \Delta x$ and employ the FA algorithm.



(a) Undeformed grid. (b) Deformed grid. (c) Deformed grid with no-fail zone.

Fig. 10: Illustration of the bond breaking in a square domain discretized with a uniform grid of grid spacing $\Delta x = \Delta y = 0.2$. The red nodes represent a single horizontal layer of nodes which is being deformed; the rest of the nodes are in blue. The list of bonds is computed based on the FA algorithm with $\delta = \Delta x$, and the resulting bonds are represented by cyan lines. In (a), the grid in the reference configuration is shown together with all bonds. In (b), the grid in the deformed configuration is shown, where only intact bonds are displayed. In (c), the same result as in (b) is presented, except that a no-fail zone is defined; the no-fail zone contains nodes being deformed having positive x-coordinates in the undeformed grid (see (a)). The bond breaking is performed with the `BondBreaking` function. The figures are produced with the `PlotBondBreaking` function.

Remark 6 Note that for the case of general grids, the FA algorithm is employed, leading to the above-described symmetry. However, for a more general case, in principle, the decision of breaking a bond connecting a given node to one of its quadrature points should be treated independently of the status of the bonds of other nodes.

3.7 `PlotField`

The function `PlotField` plots the grid nodes colored by a field value, either in the reference or current configuration depending on which nodal positions (reference or current) are provided to the function. This function can be called multiples times with different input field values to attain plots showcasing different field quantities. For instance, in Section 4.1, the function is used to plot the magnitude of the displacement field while, in Section 4.2, the function is used to plot both the macroelastic energy density and damage fields.

4 Numerical Examples

4.1 Wave propagation

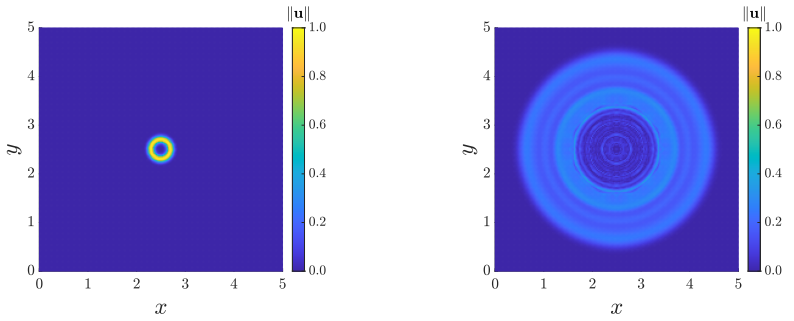
The first example demonstrates application of PDMATLAB2D to a wave propagation problem with an initial displacement based on a radial Gaussian distribution. The simulation domain is chosen as a square domain $\Omega = (0,5) \times (0,5)$, and the problem parameters are: horizon $\delta = 0.1$, mass density $\rho = 1$, and Young's modulus $E = 1$ in consistent units. We employ the GPMB PD model with the constant influence function (ω_0) under plane stress condition. No external body force density is applied, i.e., $\mathbf{b}(\mathbf{x}, t) = \mathbf{0}$. All domain boundaries are traction free, i.e., we do not explicitly apply any boundary conditions. The initial conditions are:

$$\mathbf{u}(\mathbf{x},0) = \begin{cases} A \exp\left(-\frac{(\|\mathbf{x}-\mathbf{x}_m\|-\mu)^2}{(2\sigma)^2}\right) \frac{\mathbf{x}-\mathbf{x}_m}{\|\mathbf{x}-\mathbf{x}_m\|}, & \mu - 6\sigma < \|\mathbf{x}-\mathbf{x}_m\| < \mu + 6\sigma, \\ \mathbf{0}, & \text{otherwise}, \end{cases} \tag{17}$$

$$\dot{\mathbf{u}}(\mathbf{x},0) = \mathbf{0}, \tag{18}$$

where $\mathbf{x}_m = (2.5, 2.5)$, $A = 1$, $\sigma = 1/30$, and $\mu = 6\sigma$. The magnitude of the initial displacement field is plotted in Figure 11a.

The domain is discretized with a uniform grid of 200×200 nodes, resulting in a grid spacing of $\Delta x = \Delta y = 0.025$. The FA approach (see Section 3.2.1) is employed for the computation of neighbor areas. For time integration, the velocity Verlet time-integration scheme (see Algorithm 3) is used with a time step $\Delta t = 0.01$, and the final time is $T = 1.8$, resulting in 180 time steps. Since only the dynamics of wave propagation are of concern in this example, we do not enable bond breaking. The final simulation output is shown in Figure 11b. The simulation is run with the `WavePropagation` input deck (see Section H.1).



(a) Initial configuration. (b) Final configuration.

Fig. 11: Wave propagation simulation with an initial displacement based on a radial Gaussian distribution run with the `WavePropagation` input deck.

4.2 Crack branching

The second example demonstrates application of PDMATLAB2D to a crack branching problem in a pre-notched soda-lime glass thin plate, following [52]. Due to the thinness of the plate, the problem is modeled with a two-dimensional plane stress PD model. The simulation domain is a rectangular plate with dimensions 0.1 m by 0.04 m, defined as $\Omega = (-0.05, 0.05) \times (-0.02, 0.02)$ m^2, with a vertically-centered horizontal pre-notch extending from the left boundary all the way to the center of the plate (see Figure 12a). The problem parameters are: horizon $\delta = 0.001$ m, mass density $\rho = 2440$ kg/m^3, Young's modulus $E = 72$ GPa, and fracture energy $G_0 = 3.8$ J/m^2.[5] We employ the GPMB PD model with the constant influence function (ω_0). The initial conditions are $\mathbf{u}(\mathbf{x}, 0) = \mathbf{0}$ and $\dot{\mathbf{u}}(\mathbf{x}, 0) = \mathbf{0}$. Traction boundary conditions with loading magnitude $\sigma = 2$ MPa are applied on the top and bottom boundaries, as illustrated in Figure 12a. No other body forces or boundary conditions are imposed. The traction boundary conditions are applied by prescribing a body force density, as explained below.

To convert a surface traction to a body force density, we consider a discretization in the xy-plane with grid spacings Δx and Δy in the x- and y-directions, respectively. Then, the total force magnitude in the y-direction at a node adjacent to the boundary is $|F_y^{\text{ext}}| = \sigma h \Delta x$, where $h\Delta x$ is the area on which the traction loading is applied for the corresponding cell. The magnitude of the body force density in the y-direction can be then computed as:

$$|b_y| = \frac{|F_y^{\text{ext}}|}{h\Delta x \Delta y} = \frac{\sigma}{\Delta y}, \tag{19}$$

[5]The Poisson's ratio of soda-lime glass is $\nu = 0.22$; however, the two-dimensional plane stress PD model has a Poisson's ratio of $\nu = 1/3$ [53].

where $h\Delta x\Delta y$ is the effective volume of the cell. Consequently, we prescribe a body force density $\mathbf{b}(\mathbf{x}, t) = (0, +\sigma/\Delta y)$ at any node on the top layer of nodes and $\mathbf{b}(\mathbf{x}, t) = (0, -\sigma/\Delta y)$ at any node on the bottom layer of nodes. A no-fail zone is also defined, which contains all points that are closer than δ to the top or bottom boundaries.

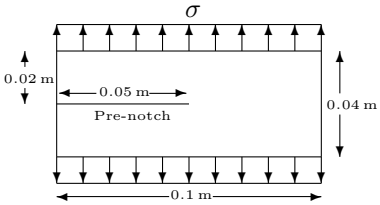
The domain is discretized with a uniform grid of 300×120 nodes, resulting in a grid spacing of $\Delta x = \Delta y \approx 3.33\text{e}-4$ m. The FA approach (see Section 3.2.1) is employed for the computation of neighbor areas. For time integration, the velocity Verlet time-integration scheme (see Algorithm 3) is used with a time step $\Delta t = 67\text{e}-3$ μs, and the final time is $T = 43$ μs, resulting in 641 time steps.

In this simulation, we enable bond breaking and compute the damage field (11). A numerical approximation of the damage at node i and time step n is given by (see Section 2.2):

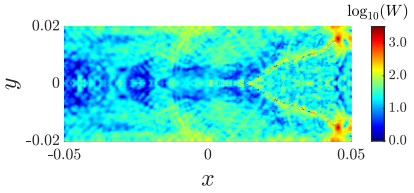
$$\varphi_i^n := 1 - \frac{\displaystyle\sum_{j \in \hat{\mathcal{F}}_i^n} \mu_{ji}^n \hat{A}_{j(i)}}{\displaystyle\sum_{j \in \hat{\mathcal{F}}_i^0} \hat{A}_{j(i)}},$$

where $\mu_{ji}^n := \mu(\hat{\mathbf{x}}_{j(i)}, \mathbf{x}_i, t^n)$, $\hat{\mathcal{F}}_i^n$ is the set of quadrature points at time step n, and $\hat{A}_{j(i)}$ is a quadrature weight; note that, due to bond breaking, $\hat{\mathcal{F}}_i^n$ may differ from $\hat{\mathcal{F}}_i^0$, which is the set of quadrature points at the initial time. We observe that, in the case of a pre-notched domain, one may compute $\hat{\mathcal{F}}_i^0$ before or after introducing pre-notches. If the computation is done prior to introducing pre-notches, then the damage field will treat pre-notches as damaged regions, regardless of whether additional bonds are broken during a simulation. In contrast, if the computation is done after introducing pre-notches, then, unless additional bonds connected to nodes adjacent to the pre-notches are broken during a simulation, the damage field will treat pre-notches as undamaged regions.

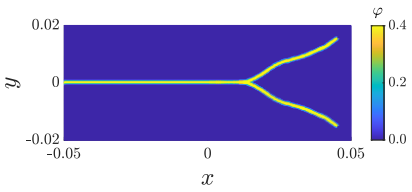
The final simulation outputs for the macroelastic energy density and damage fields are shown in Figures 12b and 12c, respectively. The simulation is run with the `CrackBranching` input deck (see Section H.2).



(a) Initial configuration.



(b) Final macroelastic energy density field.



(c) Final damage field.

Fig. 12: Crack branching simulation for a pre-notched soda-lime glass thin plate under traction loading run with the `CrackBranching` input deck.

5 Guidance for Code Extensions

There are several possible extensions to PDMATLAB2D. While some extensions are more intrusive and require an extra effort from the user, other extensions are relatively easy to implement and only require minor modifications to the code. Here, we describe several examples of relatively easy extensions with guidelines for their implementation.

5.1 Computing additional per-bond quantities

The current implementation of PDMATLAB2D computes and stores several per-bond quantities, including neighbor areas, reference lengths of neighbor bonds, coordinates of quadrature points, and influence function values within the `NeighborList` function (see Section 3.2) by calling the `NeighborAreaBondLengthCoord` function for the first three quantities and the `InfluenceFunction` function for the last one. The `NeighborList` function can be modified to compute and store additional per-bond quantities, as follows. After calling the `NeighborAreaBondLengthCoord` function, the following `if` statement is implemented:

```
% Only include neighbor if neighbor area > 0
if Vk > 0
```

```
    % Evaluate influence function
    [IFk] = InfluenceFunction(omega,rk_hat,del);
    ...
end
```

where, among other things, for a given source node, the influence function for the bond corresponding to the kth neighbor is evaluated. Subsequent lines below the call of the `InfluenceFunction` function can be added to compute other per-bond quantities. The outputs of the `NeighborList` function will then have to be modified to add the new per-bond quantities, and the inputs may need to be modified as well if additional information is required to compute the new per-bond quantities. Moreover, the function call in the `Main` script will need to be updated to reflect the change.

5.2 Using a different neighborhood shape

The current implementation of PDMATLAB2D employs a circular neighborhood (see (2)). The function `NeighborAreaBondLengthCoord` implements an `if` statement that allows selection of different algorithms, which can be leveraged to use different neighborhood shapes. As an example, if a neighborhood with a non-circular shape is desired, one could simply add an `elseif` statement to specify a condition for a new algorithm (let us call it "X" algorithm) which employs the desired neighborhood shape, as follows:

```
if strcmp(AlgName,'FA')
    % ------------------------------------------------------------------
    %                          FA algorithm
    % ------------------------------------------------------------------
    ...
elseif strcmp(AlgName,'PA-AC') ||  strcmp(AlgName,'IPA-AC')
    % ------------------------------------------------------------------
    %              PA-AC algorithm or IPA-AC algorithm
    % ------------------------------------------------------------------
    ...
elseif strcmp(AlgName,'X')
    % ------------------------------------------------------------------
    %                            X algorithm
    % ------------------------------------------------------------------
    ...
else
    error('Invalid AlgName.')
end
```

Note that, in the FA algorithm, the condition "`if r2 < del^2 + tol`" determines the shape of the circular neighborhood, where `r2` is the distance squared between a source node and a neighboring node, `del` is the PD horizon, and `tol` is a tolerance. Similarly, an alternative condition can be used for the "X" algorithm to get the desired neighborhood shape.

5.3 Changing the influence function

The current implementation of PDMATLAB2D incorporates various influence functions, specifically a constant influence function and various piecewise polynomial ones (see Section 2.1.1). Changing the influence function can be easily done, as long as the new influence function only depends on the bond length and the horizon, which are the two inputs of the `InfluenceFunction` function besides the influence function choice parameter `omega`, by adding a new case to the implemented `switch` statement. The current `switch` statement allows selection between the piecewise constant (`case` = 0.5), piecewise linear (`case` = 1), piecewise cubic (`case` = 3), piecewise quintic (`case` = 5), and piecewise septic (`case` = 7) influence functions. If a new influence function is desired (let us refer to it as the "X" influence function), one could add the case "X" with the corresponding expression for the influence function as follows:

```
% Evaluate influence function (IF)
if omega == 0
    % Constant IF
    ...
else
    ...
    switch omega
        case 0.5
            % Piecewise constant IF
            ...
        case 1
            % Piecewise linear IF
            ...
        case 3
            % Piecewise cubic IF
            ...
        case 5
            % Piecewise quintic IF
            ...
        case 7
            % Piecewise septic IF
            ...
        case 'X'
            % New influence function
            ...
        otherwise
            error('Invalid omega.');
    end
end
```

Please note that case expressions can be either numbers or strings.

Remark 7 The expressions for the micromodulus constant and the critical stretch of the GPMB model depend on the choice of influence function (see Tables 2 and 3, respectively). Therefore, implementing a new influence function for the GPMB model within the `InfluenceFunction` function requires, in addition, to implement consistent expressions for the micromodulus constant and the critical stretch in the

`PDBondConstants` function. A similar `switch` statement as the above is implemented in the `PDBondConstants` function, which can be used for this purpose.

Remark 8 For new influence functions with dependence on additional parameters, the inputs of the `InfluenceFunction` function needs to be changed to include those parameters, and the function call in the `NeighborList` function needs to be updated to reflect the addition of the new input parameters.

5.4 Changing the constitutive model

The current implementation of PDMATLAB2D only incorporates the GPMB constitutive model (see Section 2.1). Changing the constitutive model requires simultaneously changing the pairwise force function and the pairwise potential function. To this end, the `ForceEnergyDensity` function implements an `if` statement, which can be leveraged to add new constitutive models. As an example, if the "X" constitutive model is needed, one would simply add an `elseif` statement to specify a condition for the desired constitutive model, as follows:

```
if strcmp(model,'GPMB')
    ...
elseif strcmp(model,'X')
    ...
else
    error('Invalid model.')
end
```

Remark 9 Changing a constitutive model generally requires modifying the PD bond constants, which can be done by using the `PDBondConstants` function. If the new constitutive model only requires a micromodulus constant and a critical stretch, and it does not require additional input parameters for their computation, then the required constants can be added by using the same `if` statement as above, which is also implemented in the `PDBondConstants` function. However, if the new constitutive model uses other PD bond constants or requires a different set of parameters to compute the PD bond constants, the inputs or outputs of the `PDBondConstants` function need to be modify accordingly, and the function call in the `Main` script needs to be updated to reflect the change.

5.5 Changing the time-integration scheme

The current implementation of PDMATLAB2D only implements the velocity Verlet time-integration scheme (see Algorithm 3). The function `TimeIntegrator` implements an `if` statement, which can be leveraged to add new time-integration schemes. As an example, if the "X" time-integration scheme is desired, one would simply add an `elseif` statement to specify a

condition for the desired time-integration scheme, as follows:

```
if strcmp(TimeScheme,'VVerlet')
    ...
elseif strcmp(TimeScheme,'X')
    ...
else
    error('Invalid TimeScheme.')
end
```

5.6 Changing the bond-breaking criterion

The current implementation of PDMATLAB2D only incorporates the critical stretch bond-breaking criterion proposed in [26] (see Section 2.1). To include alternative bond-breaking criteria, the user can do either of the following:

- Option 1: write a modified bond-breaking function,
- Option 2: change the `BondBreaking` function,
- Option 3: add an `if` statement to the `BondBreaking` function, allowing to choose the desired bond-breaking criterion based on conditional statements, similar, for example, to the `if` statement implemented for choosing the force model in the `ForceEnergyDensity` function (see Section 5.4).

6 Summary

This paper provides an overview of PDMATLAB2D, a meshfree peridynamics implementation in MATLAB suitable for simulation of two-dimensional fracture problems. PDMATLAB2D provides an entry-level peridynamics computational tool, which can serve educational and training goals, and an accessible and easily modifiable computational environment that peridynamics researchers can adapt to simulate different peridynamics problems. The code implements a bond-based brittle elastic peridynamic model and a critical stretch criterion for bond breaking, and it is designed to be extendable for other peridynamic models and computational features. We reviewed the code structure and functions at a high level, while providing confirmations of the correctness of the computations via figures or tables. The MATLAB functions themselves are carefully commented; thus, we focused their descriptions on the rationale and implementation. Numerical examples for wave propagation and crack branching are presented to demonstrate application of PDMATLAB2D. The corresponding input decks should serve users as a reference for running alternative simulations. While the present version of the code is quite flexible and incorporates various peridynamics modeling and computational features, the addition of other components will be considered in the future, such as state-based peridynamic models, displacement and velocity boundary conditions, and contact algorithms.

Acknowledgments. Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory,

managed by UT-Battelle, LLC for the US Department of Energy under contract DE-AC05-00OR22725. This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship program.

Appendix A Derivation of a microelastic bond-based PD model

In this appendix, we derive the functional form of a microelastic bond-based PD model, following [1].

Let a bond-based PD pairwise force function be of the following form: $\mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t)$ with $\boldsymbol{\eta}, \mathbf{x}', \mathbf{x} \in \mathbb{R}^3$ and $t \geqslant 0$, where $\boldsymbol{\eta} := \mathbf{u}(\mathbf{x}', t) - \mathbf{u}(\mathbf{x}, t)$. Then, the *linear admissibility condition* implies:

$$\mathbf{f}(-\boldsymbol{\eta}, \mathbf{x}, \mathbf{x}', t) = \mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t), \qquad \forall \boldsymbol{\eta}, \mathbf{x}', \mathbf{x} \in \mathbb{R}^3, \quad t \geqslant 0, \tag{A1}$$

and the *angular admissibility condition* implies:

$$(\boldsymbol{\xi} + \boldsymbol{\eta}) \times \mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = \mathbf{0}, \qquad \forall \boldsymbol{\eta}, \mathbf{x}', \mathbf{x} \in \mathbb{R}^3, \quad t \geqslant 0, \tag{A2}$$

where $\boldsymbol{\xi} := \mathbf{x}' - \mathbf{x}$. By (A1) and (A2), we can express the pairwise force function as follows:

$$\mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = F(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t)(\boldsymbol{\xi} + \boldsymbol{\eta}), \tag{A3}$$

where F is a symmetric scalar-valued function:

$$F(-\boldsymbol{\eta}, \mathbf{x}, \mathbf{x}', t) = F(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t), \qquad \forall \boldsymbol{\eta}, \mathbf{x}', \mathbf{x} \in \mathbb{R}^3, \quad t \geqslant 0. \tag{A4}$$

For a microelastic material, the following condition holds:

$$\nabla_{\boldsymbol{\eta}} \times \mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = \mathbf{0}, \qquad \forall \boldsymbol{\eta}, \mathbf{x}', \mathbf{x} \in \mathbb{R}^3, \quad t \geqslant 0. \tag{A5}$$

Assuming F is continuously differentiable on $\boldsymbol{\eta}$, we have:

$$\begin{aligned}
\nabla_{\boldsymbol{\eta}} \times \mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) &= \varepsilon_{ijk} \frac{\partial}{\partial \eta_i} \left(f_j(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) \right) \mathbf{e}_k \\
&= \varepsilon_{ijk} \frac{\partial}{\partial \eta_i} \left(F(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t)(\xi_j + \eta_j) \right) \mathbf{e}_k \\
&= \varepsilon_{ijk} \frac{\partial F}{\partial \eta_i} (\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t)(\xi_j + \eta_j) \mathbf{e}_k \\
&= \frac{\partial F}{\partial \boldsymbol{\eta}} (\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) \times (\boldsymbol{\xi} + \boldsymbol{\eta}),
\end{aligned}$$

where ε_{ijk} is the Levi-Civita symbol, $\{\mathbf{e}_k\}_{k=1,2,3}$ is the set of standard Cartesian unit vectors, and we used Einstein summation convention for repeated

indices. Then, by (A5), we have

$$\frac{\partial F}{\partial \boldsymbol{\eta}}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = A(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t)(\boldsymbol{\xi} + \boldsymbol{\eta}),$$

where A is a scalar-valued function. Note that

$$\frac{\partial}{\partial \eta_i}(\|\boldsymbol{\xi} + \boldsymbol{\eta}\|) = \frac{\xi_i + \eta_i}{\|\boldsymbol{\xi} + \boldsymbol{\eta}\|}, i = 1, 2, 3.$$

Consequently, we can express F as follows:

$$F(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = H(\|\boldsymbol{\xi} + \boldsymbol{\eta}\|, \mathbf{x}', \mathbf{x}, t), \tag{A6}$$

where $H(p, \mathbf{x}', \mathbf{x}, t)$ with $p = \|\boldsymbol{\xi} + \boldsymbol{\eta}\|$ is a scalar-valued function continuously differentiable on p.

Combining (A6) and (A3), we can express the microelastic bond-based PD pairwise force function as follows:

$$\mathbf{f}(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}, t) = f(\|\boldsymbol{\xi} + \boldsymbol{\eta}\|, \mathbf{x}', \mathbf{x}, t)\frac{\boldsymbol{\xi} + \boldsymbol{\eta}}{\|\boldsymbol{\xi} + \boldsymbol{\eta}\|}, \tag{A7}$$

where f is a scalar-valued function given by

$$f(\|\boldsymbol{\xi} + \boldsymbol{\eta}\|, \mathbf{x}', \mathbf{x}, t) = \|\boldsymbol{\xi} + \boldsymbol{\eta}\| H(\|\boldsymbol{\xi} + \boldsymbol{\eta}\|, \mathbf{x}', \mathbf{x}, t).$$

Appendix B Derivation of maximum perturbation in `GridGenerator`

Consider a rectangular cell centered at node i with coordinates (x_i, y_i) with edge lengths Δx and Δy in the x- and y-directions, respectively, and assume a perturbation of the cell vertices. For the purpose of the derivations below, we assume only three of the four vertices are perturbed, and we denote these by A, B, and C, as illustrated in Figure B1. The perturbed vertices are denoted by A', B', and C', respectively (see Figure B1). Each of the vertices is independently perturbed, and the region of admissible new positions for a given vertex after perturbation is given by a rectangular region of size $2\alpha\Delta x \times 2\alpha\Delta y$ centered at the vertex (see Figure B1), where α is a given parameter. We seek the value of the parameter α for which C' falls on the line connecting A' and B' (see Figure B1), which represents the limiting value for which the shape of the deformed cell remains convex.

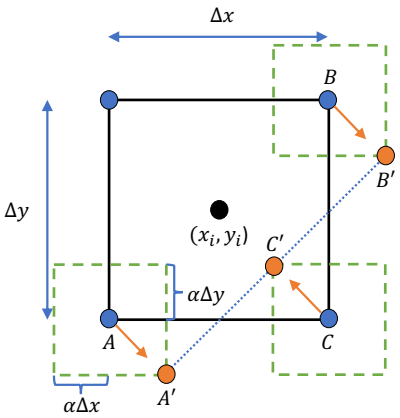


Fig. B1: Representation of perturbation of the vertices of a cell centered at node i with coordinates (x_i, y_i). The cell dimensions are Δx and Δy in the x- and y-directions, respectively. The unperturbed cell vertices are represented by blue circles, whereas the perturbed cell vertices are represented by orange circles. Only the vertices A, B, and C are perturbed, and their corresponding vertices after perturbation are A', B', and C'. The region of perturbation for each of these vertices is represented by an empty rectangle with green dashed border line. The size of the rectangle is defined as a fraction of the original cell via a parameter α.

Consider the following three vertices:

$$\begin{aligned} A &:= (x_i - \tfrac{\Delta x}{2}, y_i - \tfrac{\Delta y}{2}), \\ B &:= (x_i + \tfrac{\Delta x}{2}, y_i + \tfrac{\Delta y}{2}), \\ C &:= (x_i + \tfrac{\Delta x}{2}, y_i - \tfrac{\Delta y}{2}). \end{aligned}$$

To consider the worst case scenario, we assume the following perturbed vertices (see Figure B1):

$$\begin{aligned} A' &:= (x_i - \tfrac{\Delta x}{2} + \alpha\Delta x, y_i - \tfrac{\Delta y}{2} - \alpha\Delta y), \\ B' &:= (x_i + \tfrac{\Delta x}{2} + \alpha\Delta x, y_i + \tfrac{\Delta y}{2} - \alpha\Delta y), \\ C' &:= (x_i + \tfrac{\Delta x}{2} - \alpha\Delta x, y_i - \tfrac{\Delta y}{2} + \alpha\Delta y). \end{aligned}$$

Let the line connecting the nodes A' and B' be

$$y(x) = a_1 x + a_0, \tag{B8}$$

where a_1 and a_0 are parameters. The slope of the line, a_1, is given by

$$a_1 = \frac{\Delta y}{\Delta x}. \tag{B9}$$

Substituting a_1 in (B8), and evaluating the line for the vertex A', we have

$$y_i - \tfrac{\Delta y}{2} - \alpha\Delta y = \frac{\Delta y}{\Delta x}(x_i - \tfrac{\Delta x}{2} + \alpha\Delta x) + a_0,$$

from which a_0 can be obtained:

$$a_0 = y_i - \frac{\Delta y}{\Delta x}x_i - 2\alpha\Delta y. \tag{B10}$$

We now substitute the coordinates of C' in the line given by (B8) with coefficients defined by (B9) and (B10):

$$\begin{aligned} y_i - \tfrac{\Delta y}{2} + \alpha\Delta y &= a_1(x_i + \tfrac{\Delta x}{2} - \alpha\Delta x) + a_0 \\ &= \frac{\Delta y}{\Delta x}(x_i + \tfrac{\Delta x}{2} - \alpha\Delta x) + y_i - \frac{\Delta y}{\Delta x}x_i - 2\alpha\Delta y, \end{aligned}$$

from which we obtain, after some algebraic manipulations,

$$\alpha = \frac{1}{4}.$$

Appendix C Derivation of c and s_0

This appendix presents derivations of the micromodulus constant, c, and critical stretch, s_0, for the GPMB model (see Section 2.1) in two dimensions, following analogous derivations for the PMB model in three dimensions [26].

C.1 Micromodulus constant

The micromodulus constant, c, can be obtained by equating the PD macroelastic energy density and the strain energy density from the classical theory of elasticity, under a suitable deformation.

Consider a static isotropic extension given by $\mathbf{u}(\mathbf{x}) = \bar{s}\mathbf{x}$ with $\bar{s}$ constant. In this case, $\boldsymbol{\eta} = \bar{s}\boldsymbol{\xi}$, which results in a constant stretch: $s(\|\boldsymbol{\xi} + \boldsymbol{\eta}\|, \|\boldsymbol{\xi}\|) = \bar{s}$ for all $\boldsymbol{\eta}, \boldsymbol{\xi} \in \mathbb{R}^2$. Then, we can express the macroelastic energy density of the GPMB model for a point $\mathbf{x}$ in the bulk of a body (assuming $\mu \equiv 1$) as follows (see (7) and (10)):

$$W = \frac{1}{4}h\int_{\mathcal{H}} c\,\omega(\|\boldsymbol{\xi}\|)\,\bar{s}^2\|\boldsymbol{\xi}\|d\boldsymbol{\xi},$$

where we used the change of variable $\mathbf{x}' = \mathbf{x} + \boldsymbol{\xi}$ and

$$\mathcal{H} := \{\boldsymbol{\xi} \in \mathbb{R}^2 : \|\boldsymbol{\xi}\| \leqslant \delta\}$$

is the neighborhood around the origin (*cf.* (2)). Using polar coordinates,

$$W = \frac{1}{4} ch\bar{s}^2 \int_0^{2\pi} \int_0^{\delta} \omega(r) r \, r dr d\theta = c\frac{\pi}{2} h \left(\int_0^{\delta} \omega(r) r^2 dr \right) \bar{s}^2. \tag{C11}$$

Note that the arguments of W have been omitted. The time dependence has been omitted because the imposed deformation is static; the spatial dependence, on the other hand, has been omitted because the macroelastic energy density is spatially independent in the bulk of a body for an isotropic extension. Below, we similarly omit these arguments for clarity.

The integral in (C11) can be computed by direct computation for the influence functions presented in Section 2.1.1, resulting in the following expressions:

$$\int_0^{\delta} \omega_0(r) r^2 dr = \frac{\delta^3}{3}, \quad \int_0^{\delta} \omega_1(r) r^2 dr = \frac{\delta^3}{12}, \quad \int_0^{\delta} \omega_3(r) r^2 dr = \frac{\delta^3}{15}, \tag{C12}$$
$$\int_0^{\delta} \omega_5(r) r^2 dr = \frac{5}{84}\delta^3, \quad \int_0^{\delta} \omega_7(r) r^2 dr = \frac{1}{18}\delta^3.$$

Plane strain: In classical linear elasticity, the strain energy density for an isotropic material in a state of *plane strain* is given by (see, e.g. [53]):

$$W^{C\varepsilon} = \frac{E}{2(1+\nu)(1-2\nu)} \left[(1-\nu)(\varepsilon_{11}^2 + \varepsilon_{22}^2) + 2\nu\varepsilon_{11}\varepsilon_{22} + 2(1-2\nu)\varepsilon_{12} \right], \tag{C13}$$

where E is the Young's modulus, ν is the Poisson's ratio, and ε_{ij} with $i, j = 1, 2$ are the components of the infinitesimal strain tensor:

$$\varepsilon_{ij} := \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right), \quad i, j = 1, 2.$$

For the isotropic extension given by $\mathbf{u}(\mathbf{x}) = \bar{s}\mathbf{x}$, we have $\varepsilon_{11} = \varepsilon_{22} = \bar{s}$ and $\varepsilon_{12} = 0$. Then,

$$W^{C\varepsilon} = \frac{E}{2(1+\nu)(1-2\nu)} \left[(1-\nu)(\bar{s}^2 + \bar{s}^2) + 2\nu\bar{s}^2 \right] = \frac{E\bar{s}^2}{(1+\nu)(1-2\nu)}.$$

In the case of plane strain, two-dimensional bond-based PD models are restricted to a Poisson's ratio of $\nu = 1/4$ [38, 53]. In this case,

$$W^{C\varepsilon} = \frac{8}{5} E \bar{s}^2. \tag{C14}$$

Assuming $W = W^{C\varepsilon}$, we thus obtain from (C11) and (C14):

$$c = \frac{16}{5\pi} \frac{E}{h \int_0^\delta \omega(r) r^2 dr}, \tag{C15}$$

where the result depends on the choice of influence function. Using (C12), the resulting expressions corresponding to the different influence functions are summarized in Section 3.4 (see Table 2 (top row)).

Plane stress: In classical linear elasticity, the strain energy density for an isotropic material in a state of *plane stress* is given by (see, e.g. [53]):

$$W^{C\sigma} = \frac{E}{2(1-\nu^2)} \left[\varepsilon_{11}^2 + \varepsilon_{22}^2 + 2\nu \varepsilon_{11} \varepsilon_{22} + 2(1-\nu)\varepsilon_{12} \right]. \tag{C16}$$

For the isotropic extension given by $\mathbf{u}(\mathbf{x}) = \bar{s}\mathbf{x}$, we have (recall $\varepsilon_{11} = \varepsilon_{22} = \bar{s}$ and $\varepsilon_{12} = 0$)

$$W^{C\sigma} = \frac{E}{2(1-\nu^2)} \left[\bar{s}^2 + \bar{s}^2 + 2\nu \bar{s}^2 \right] = \frac{E \bar{s}^2}{(1-\nu)}.$$

In the case of plane stress, two-dimensional bond-based PD models are restricted to a Poisson's ratio of $\nu = 1/3$ [38, 53]. In this case,

$$W^{C\sigma} = \frac{3}{2} E \bar{s}^2. \tag{C17}$$

Assuming $W = W^{C\sigma}$, we thus obtain from (C11) and (C17):

$$c = \frac{3}{\pi} \frac{E}{h \int_0^\delta \omega(r) r^2 dr}, \tag{C18}$$

where the result again depends on the choice of influence function. Using (C12), the resulting expressions corresponding to the different influence functions are summarized in Section 3.4 (see Table 2 (bottom row)).

Remark 10 We observe that, while the expressions for the micromodulus constant c in (C15) and (C18) depend on h, in practice, h cancels out when c is substituted in expressions for the computation of PD quantities (see, e.g., (1)).

C.2 Critical stretch

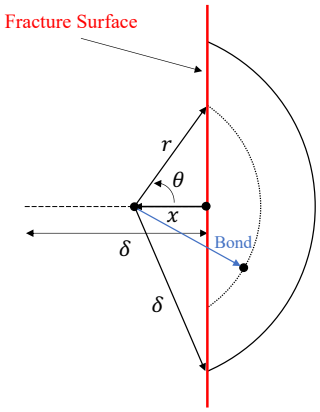


Fig. C2: Illustration of the integration for computation of the fracture energy G_0 in (C19) in two dimensions, analogously to [26]. Given a fracture surface, we compute the energy per unit area collectively provided by the pairwise potentials of all the bonds connecting any point on the left side of the fracture surface along a line perpendicular to the surface with distance $0 < x \leqslant \delta$ to points within their neighborhood on the right side of the fracture surface.

To compute the critical stretch, s_0, we assume any bond breaks irreversibly when its stretch exceeds the critical stretch. Then, we consider a fracture surface and compute the total energy per unit area released by breaking all bonds crossing that surface and compare the result to the fracture energy, G_0.

Mimicking the calculations in [26], adapted to a two-dimensional case, we have (*cf.* (10))

$$G_0 = h \int_0^{\delta} \int_x^{\delta} \int_{-\cos^{-1}(x/r)}^{\cos^{-1}(x/r)} \left(\frac{1}{2} c\omega(r) s_0^2 r \right) r d\theta dr dx. \tag{C19}$$

We now have

$$G_0 = \frac{1}{2} chs_0^2 \int_0^{\delta} \int_x^{\delta} \omega(r) r^2 2\cos^{-1}(x/r) dr dx.$$

Changing the order of integration:

$$G_0 = chs_0^2 \int_0^{\delta} \int_0^{r} \omega(r) r^2 \cos^{-1}(x/r) dx dr.$$

Using the change of variable $a = x/r$, we have

$$G_0 = chs_0^2 \int_0^\delta \int_0^1 \omega(r) r^2 \cos^{-1}(a) r da dr = chs_0^2 \int_0^\delta \omega(r) r^3 \left[\int_0^1 \cos^{-1}(a) da \right] dr$$
$$= chs_0^2 \int_0^\delta \omega(r) r^3 dr,$$

where

$$\int_0^1 \cos^{-1}(a) da = 1.$$

The value of s_0 can be then expressed as follows:

$$s_0 = \sqrt{\frac{G_0}{ch \int_0^\delta \omega(r) r^3 dr}}. \tag{C20}$$

The integral in (C20) can be computed by direct computation for the influence functions presented in Section 2.1.1, resulting in the following expressions:

$$\int_0^\delta \omega_0(r) r^3 dr = \frac{\delta^4}{4}, \quad \int_0^\delta \omega_1(r) r^3 dr = \frac{\delta^4}{20}, \quad \int_0^\delta \omega_3(r) r^3 dr = \frac{\delta^4}{28}, \tag{C21}$$
$$\int_0^\delta \omega_5(r) r^3 dr = \frac{5}{168}\delta^4, \quad \int_0^\delta \omega_7(r) r^3 dr = \frac{7}{264}\delta^4.$$

Using these expressions as well as the expressions for the micromodulus constant c from Section 3.4 (see Table 2) in (C20), the resulting expressions for the critical stretch corresponding to the different influence functions are summarized in Section 3.4 (see Table 3) for both plane strain (top row) and plane stress (bottom row).

Appendix D Quadratic deformation test for the `ForceEnergyDensity` function

Consider a quadratic static deformation given by a displacement field $\mathbf{u}(x, y) = (v(x, y), w(x, y))$ of the following form:

$$v(x, y) = V_{11} x^2, \tag{D22a}$$
$$w(x, y) = 0, \tag{D22b}$$

where V_{11} is a constant. We would like to evaluate the internal force density and macroelastic energy density at a point $\mathbf{x}$ in the bulk of a body for the GPMB model. For this purpose, we approximate these quantities by resorting

to their linearization under the assumption that $\|\boldsymbol{\eta}\| \ll \delta$. In this case, the stretch in (9) can be approximated via a Taylor expansion, as follows:

$$s(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|) = s(\|\boldsymbol{\xi}\|,\|\boldsymbol{\xi}\|) + \left.\frac{\partial s(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|)}{\partial \eta_i}\right|_{\boldsymbol{\eta}=\mathbf{0}} \eta_i + \mathcal{O}(\|\boldsymbol{\eta}\|^2), \quad \text{(D23)}$$

where we used Einstein summation convention for repeated indices, and we note that $s(\|\boldsymbol{\xi}\|,\|\boldsymbol{\xi}\|) = 0$ for all $\boldsymbol{\xi} \in \mathbb{R}^2$ (see (9)). Expressing the stretch as follows:

$$s(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|) = \frac{1}{\|\boldsymbol{\xi}\|}\sqrt{(\xi_1+\eta_1)^2+(\xi_2+\eta_2)^2} - 1,$$

where $\boldsymbol{\xi} = (\xi_1,\xi_2)$ and $\boldsymbol{\eta} = (\eta_1,\eta_2)$, we can easily compute

$$\frac{\partial s(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|)}{\partial \eta_i} = \frac{1}{\|\boldsymbol{\xi}\|}\frac{(\xi_i+\eta_i)}{\|\boldsymbol{\xi}+\boldsymbol{\eta}\|}, \quad i=1,2,$$

which results in (see (D23))

$$s(\|\boldsymbol{\xi}+\boldsymbol{\eta}\|,\|\boldsymbol{\xi}\|) \approx \frac{\boldsymbol{\xi}\cdot\boldsymbol{\eta}}{\|\boldsymbol{\xi}\|^2},$$

where we omitted terms of order $\mathcal{O}(\|\boldsymbol{\eta}\|^2)$. The GPMB pairwise force function (6) with (8) and pairwise potential function (*cf.* (10)) can be then approximated (assuming $\mu \equiv 1$ and omitting the time dependence) as follows:

$$\mathbf{f}(\boldsymbol{\eta},\mathbf{x}',\mathbf{x}) \approx c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3}(\boldsymbol{\xi}\cdot\boldsymbol{\eta})\boldsymbol{\xi}, \quad \text{(D24)}$$

$$w(\boldsymbol{\eta},\mathbf{x}',\mathbf{x}) \approx \frac{1}{2}c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3}(\boldsymbol{\xi}\cdot\boldsymbol{\eta})^2. \quad \text{(D25)}$$

Using the linearized expression for the pairwise force function and pairwise potential function in (D24) and (D25), respectively, we compute the internal force density and macroelastic energy density for a point $\mathbf{x}$ in the bulk of a body under the quadratic deformation in (D22). To this end, we first note that, in this case, the relative displacement is given by

$$\begin{aligned}\boldsymbol{\eta} &= (V_{11}(x')^2 - V_{11}x^2, 0) = (V_{11}(x+\xi_1)^2 - V_{11}x^2, 0)\\ &= (V_{11}(x^2+2x\xi_1+\xi_1^2) - V_{11}x^2, 0,) = (V_{11}(2x\xi_1+\xi_1^2), 0).\end{aligned}$$

We then obtain,

$$\boldsymbol{\xi}\cdot\boldsymbol{\eta} = (\xi_1,\xi_2)\cdot(V_{11}(2x\xi_1+\xi_1^2),0) = V_{11}(2x\xi_1^2+\xi_1^3).$$

Then,

$$f_1(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}) \approx c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3} V_{11}(2x\xi_1^3 + \xi_1^4),$$
$$f_2(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}) \approx c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3} V_{11}(2x\xi_1^2 + \xi_1^3)\xi_2,$$
$$w(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x}) \approx \frac{1}{2} c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3} V_{11}^2(4x^2\xi_1^4 + 4x\xi_1^5 + \xi_1^6).$$

The components of the internal force density at $\mathbf{x}$ are given by:

$$\begin{aligned} F_1^{\text{int}}(\mathbf{x}) &:= h\int_{\mathcal{H}_\mathbf{x}} f_1(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x})d\mathbf{x}' \approx h\int_{\mathcal{H}_\mathbf{x}} c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3} V_{11}(2x\xi_1^3 + \xi_1^4)d\mathbf{x}' \\ &= h\int_{\mathcal{H}_\mathbf{x}} c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3} V_{11}\xi_1^4 d\mathbf{x}' = h\int_0^{2\pi}\int_0^{\delta} c\,\omega(r)\,\frac{1}{r^3} V_{11}(r\cos(\theta))^4 r\,dr\,d\theta \\ &= \frac{3\pi}{4} chV_{11}\int_0^{\delta} \omega(r) r^2 dr, \\ F_2^{\text{int}}(\mathbf{x}) &:= h\int_{\mathcal{H}_\mathbf{x}} f_2(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x})d\mathbf{x}' \approx 0, \end{aligned}$$

where we used antisymmetry considerations to eliminate the term with ξ_1^3 in the expression for $F_1^{\text{int}}(\mathbf{x})$ and to conclude that $F_2^{\text{int}}(\mathbf{x}) \approx 0$. Using the expressions for c for plane strain (C15) and plane stress (C18), we obtain

$$F_1^{\text{int}}(\mathbf{x}) \approx \begin{cases} \frac{12}{5} V_{11} E, & \text{Plane strain,} \\ \frac{9}{4} V_{11} E, & \text{Plane stress,} \end{cases} \tag{D26a}$$

$$F_2^{\text{int}}(\mathbf{x}) \approx 0. \tag{D26b}$$

We now compute the macroelastic energy density at $\mathbf{x}$:

$$\begin{aligned} W(\mathbf{x}) &:= \frac{1}{2} h\int_{\mathcal{H}_\mathbf{x}} w(\boldsymbol{\eta}, \mathbf{x}', \mathbf{x})d\mathbf{x}' \approx \frac{1}{4} h\int_{\mathcal{H}_\mathbf{x}} c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3} V_{11}^2(4x^2\xi_1^4 + 4x\xi_1^5 + \xi_1^6)d\mathbf{x}' \\ &= \frac{1}{4} h\int_{\mathcal{H}_\mathbf{x}} c\,\omega(\|\boldsymbol{\xi}\|)\,\frac{1}{\|\boldsymbol{\xi}\|^3} V_{11}^2(4x^2\xi_1^4 + \xi_1^6)d\mathbf{x}' \\ &= \frac{1}{4} h\int_0^{2\pi}\int_0^{\delta} c\,\omega(r)\,\frac{1}{r^3} V_{11}^2(4x^2(r\cos(\theta))^4 + (r\cos(\theta))^6) r\,dr\,d\theta \\ &= \frac{1}{4} chV_{11}^2\int_0^{\delta} \omega(r)\,\frac{1}{r^3}\left(4x^2\frac{3\pi}{4}r^4 + \frac{5\pi}{8}r^6\right) r\,dr \\ &= \frac{\pi}{32} chV_{11}^2\int_0^{\delta} \omega(r)\,\left(24x^2r^2 + 5r^4\right) dr. \end{aligned}$$

	ω_0	ω_1	ω_3	ω_5	ω_7
Plane strain W	$\frac{EV_{11}^2}{10}\left[24x^2+3\delta^2\right]$	$\frac{EV_{11}^2}{10}\left[24x^2+2\delta^2\right]$	$\frac{EV_{11}^2}{10}\left[24x^2+\frac{45\delta^2}{28}\right]$	$\frac{EV_{11}^2}{10}\left[24x^2+\frac{7\delta^2}{5}\right]$	$\frac{EV_{11}^2}{10}\left[24x^2+\frac{14\delta^2}{11}\right]$
Plane stress W	$\frac{3EV_{11}^2}{32}\left[24x^2+3\delta^2\right]$	$\frac{3EV_{11}^2}{32}\left[24x^2+2\delta^2\right]$	$\frac{3EV_{11}^2}{32}\left[24x^2+\frac{45\delta^2}{28}\right]$	$\frac{3EV_{11}^2}{32}\left[24x^2+\frac{7\delta^2}{5}\right]$	$\frac{3EV_{11}^2}{32}\left[24x^2+\frac{14\delta^2}{11}\right]$

Table D1: Macroelastic energy density for a point $\mathbf{x}=(x,y)$ in the bulk of a body under the quadratic deformation given in (D22).

Using the expressions for c for plane strain (C15) and plane stress (C18), we can express the macroelastic energy density, as follows:

- Plane strain

$$W(\mathbf{x}) \approx \frac{\pi}{32}\frac{16}{5\pi}\frac{E}{h\displaystyle\int_0^\delta \omega(r)r^2dr}hV_{11}^2\int_0^\delta \omega(r)\,\left(24x^2r^2+5r^4\right)dr$$
$$= \frac{EV_{11}^2}{10}\left[24x^2+5\,\frac{\displaystyle\int_0^\delta \omega(r)r^4dr}{\displaystyle\int_0^\delta \omega(r)r^2dr}\right]. \tag{D27}$$

- Plane stress

$$W(\mathbf{x}) \approx \frac{\pi}{32}\frac{3}{\pi}\frac{E}{h\displaystyle\int_0^\delta \omega(r)r^2dr}hV_{11}^2\int_0^\delta \omega(r)\,\left(24x^2r^2+5r^4\right)dr$$
$$= \frac{3EV_{11}^2}{32}\left[24x^2+5\,\frac{\displaystyle\int_0^\delta \omega(r)\,r^4dr}{\displaystyle\int_0^\delta \omega(r)r^2dr}\right]. \tag{D28}$$

We now have

$$\int_0^\delta \omega_0(r)r^4dr=\frac{\delta^5}{5},\quad \int_0^\delta \omega_1(r)r^4dr=\frac{\delta^5}{30},\quad \int_0^\delta \omega_3(r)r^4dr=\frac{3\delta^5}{140},\tag{D29}$$
$$\int_0^\delta \omega_5(r)r^4dr=\frac{\delta^5}{60},\quad \int_0^\delta \omega_7(r)r^4dr=\frac{7\delta^5}{495}.$$

Using the expressions in (C12) and (D29) in (D27) and (D28), we obtain the results reported in Table D1.

Remark 11 Given the quadratic deformation in (D22), the strain components are: $\varepsilon_{11}=2V_{11}x$, $\varepsilon_{12}=0$, and $\varepsilon_{22}=0$. Consequently, the corresponding strain energy

density in classical linear elasticity, for the case of plane strain ($\nu = 1/4$), is (see (C13))

$$W^{C\varepsilon} = \frac{E(1-\nu)}{2(1+\nu)(1-2\nu)}(2V_{11}x)^2 = \frac{12}{5}EV_{11}x^2$$

and, for the case of plane stress ($\nu = 1/3$), is (see (C16))

$$W^{C\sigma} = \frac{E}{2(1-\nu^2)}(2V_{11}x)^2 = \frac{9}{4}EV_{11}x^2,$$

which correspond to the expressions reported in Table D1, in the limit as $\delta \to 0$.

Remark 12 The stress-strain relation in plane-strain classical linear elasticity is given by:

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{12}\end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix}1-\nu & \nu & 0\\ \nu & 1-\nu & 0\\ 0 & 0 & \frac{1}{2}(1-2\nu)\end{bmatrix}\begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ 2\varepsilon_{12}\end{bmatrix}, \tag{D30}$$

where σ_{ij}, $i,j = 1,2$, are the components of the stress tensor and ε_{ij}, $i,j = 1,2$, are the components of the strain tensor, and where E is the Young's modulus and ν is the Poisson's ratio. Under the quadratic deformation given in (D22), $\varepsilon_{11} = 2V_{11}x$, $\varepsilon_{12} = 0$, and $\varepsilon_{22} = 0$, and we get (for $\nu = 1/4$):

$$\begin{aligned}\sigma_{11} &= \frac{12}{5}EV_{11}x,\\ \sigma_{22} &= \frac{4}{5}EV_{11}x,\\ \sigma_{12} &= 0.\end{aligned}$$

The internal force density is given by $\mathbf{F}^{\mathrm{CE}} := \nabla \cdot \boldsymbol{\sigma}$, which, in component form in this case, is:

$$\begin{aligned}F_1^{\mathrm{CE}} &= \frac{\partial \sigma_{11}}{\partial x} + \frac{\partial \sigma_{12}}{\partial y} = \frac{12}{5}V_{11}E,\\ F_2^{\mathrm{CE}} &= \frac{\partial \sigma_{12}}{\partial x} + \frac{\partial \sigma_{22}}{\partial y} = 0.\end{aligned}$$

Similarly, the stress-strain relation in plane-stress classical linear elasticity is given by:

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{12}\end{bmatrix} = \frac{E}{(1-\nu^2)}\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \frac{1}{2}(1-\nu)\end{bmatrix}\begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ 2\varepsilon_{12}\end{bmatrix}. \tag{D31}$$

Under the quadratic deformation given in (D22), $\varepsilon_{11} = 2V_{11}x$, $\varepsilon_{12} = 0$, and $\varepsilon_{22} = 0$, and we get (for $\nu = 1/3$):

$$\begin{aligned}\sigma_{11} &= \frac{9}{4}EV_{11}x,\\ \sigma_{22} &= \frac{3}{4}EV_{11}x,\\ \sigma_{12} &= 0.\end{aligned}$$

The internal force density, in component form in this case, is:

$$\begin{aligned}F_1^{\mathrm{CE}} &= \frac{9}{4}V_{11}E,\\ F_2^{\mathrm{CE}} &= 0.\end{aligned}$$

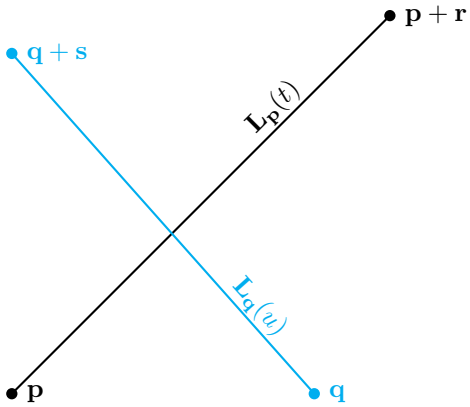


Fig. E3: Illustration of line segments $\mathbf{L}_\mathbf{p}(t)$ with endpoints $\mathbf{p}$ and $\mathbf{p}+\mathbf{r}$ and $\mathbf{L}_\mathbf{q}(u)$ with endpoints $\mathbf{q}$ and $\mathbf{q}+\mathbf{s}$.

The results for both plane strain and plane stress coincide with the ones in (D26); i.e., for the given quadratic deformation, the linearized GPMB model possesses the same internal force density (for points in the bulk of a body) as in classical linear elasticity.

Appendix E Intersection of two line segments

This section describes a method for determining if two line segments in a plane intersect. It is a planar treatment of the line segment intersection algorithm found in [54].

Consider two line segments in the xy-plane. Without loss of generality, one may select points $\mathbf{p}$ and $\mathbf{q}$ and directions $\mathbf{r}$ and $\mathbf{s}$ so that the line segments are defined parametrically as:

$$\mathbf{L}_\mathbf{p}(t) = \mathbf{p} + t\mathbf{r}, \quad t \in [0,1] \tag{E32}$$

and

$$\mathbf{L}_\mathbf{q}(u) = \mathbf{q} + u\mathbf{s}, \quad u \in [0,1]. \tag{E33}$$

An illustration is shown in Figure E3.

We start by determining if the two line segments are parallel, which is easily determined by calculating the cross product $\mathbf{r}\times\mathbf{s}$. Specifically, $\mathbf{r}\times\mathbf{s} = \mathbf{0}$ implies the segments are parallel and $\mathbf{r}\times\mathbf{s} \neq \mathbf{0}$ implies the segments are not parallel. The two cases are treated separately, starting with parallel line segments.

Line segments are parallel ***($\mathbf{r}\times\mathbf{s} = \mathbf{0}$):***

The next step is to determine if the line segments are collinear, which is true if and only if $\mathbf{q}-\mathbf{p}$ is parallel to $\mathbf{r}$, i.e., $(\mathbf{p}-\mathbf{q})\times\mathbf{r} = \mathbf{0}$. If the segments are not collinear, they clearly do not intersect. On the other hand, if the segments are

collinear, one must check whether the segments overlap. This is accomplished by writing the endpoints $\mathbf{p}$ and $\mathbf{p}+\mathbf{r}$ of the line segment $\mathbf{L_p}$ in terms of the line segment $\mathbf{L_q}$:

$$\mathbf{p} = \mathbf{q} + u_0\mathbf{s} \quad \text{and} \quad \mathbf{p}+\mathbf{r} = \mathbf{q} + u_1\mathbf{s}. \tag{E34}$$

Solving for u_0 and u_1 in (E34) yields

$$u_0 = \frac{(\mathbf{p}-\mathbf{q})\cdot\mathbf{s}}{\mathbf{s}\cdot\mathbf{s}} \quad \text{and} \quad u_1 = \frac{(\mathbf{p}+\mathbf{r}-\mathbf{q})\cdot\mathbf{s}}{\mathbf{s}\cdot\mathbf{s}} = u_0 + \frac{\mathbf{r}\cdot\mathbf{s}}{\mathbf{s}\cdot\mathbf{s}}. \tag{E35}$$

There is no overlap if and only if $\max\{u_0, u_1\} < 0$ or $\min\{u_0, u_1\} > 1$. Geometrically, these two conditions imply that along the line containing the two line segments, $\mathbf{L_p}$ is entirely to the left of $\mathbf{L_q}$ or $\mathbf{L_p}$ is entirely to the right of $\mathbf{L_q}$. Next, line segments which are not parallel are considered.

Line segments are not parallel $(\mathbf{r}\times\mathbf{s}) \neq \mathbf{0}$***:***

The line segments intersect if there exist u_0 and t_0 in the interval $[0,1]$ such that

$$\mathbf{L_p}(t_0) = \mathbf{L_q}(u_0) \Rightarrow \mathbf{p} + t_0\mathbf{r} = \mathbf{q} + u_0\mathbf{s}. \tag{E36}$$

This may be determined by solving for t_0 and u_0. In (E36), subtracting $\mathbf{p}$ and then taking the cross product with respect to $\mathbf{s}$ results in (note $\mathbf{s}\times\mathbf{s} = \mathbf{0}$)

$$t_0(\mathbf{r}\times\mathbf{s}) = (\mathbf{q}-\mathbf{p})\times\mathbf{s}. \tag{E37}$$

Since $\mathbf{r}, \mathbf{s}$, and $\mathbf{q}-\mathbf{p}$ are in the xy-plane, the first two components of the cross products in (E37) are zero. We conclude

$$t_0 = \frac{((\mathbf{q}-\mathbf{p})\times\mathbf{s})_z}{(\mathbf{r}\times\mathbf{s})_z}, \tag{E38}$$

where the subscript z signifies the third component of the vector. Similarly, in (E36) subtracting $\mathbf{q}$ and then taking the cross product with respect to $\mathbf{r}$ results in (note $\mathbf{r}\times\mathbf{r} = \mathbf{0}$)

$$u_0(\mathbf{s}\times\mathbf{r}) = (\mathbf{p}-\mathbf{q})\times\mathbf{r} \Rightarrow u_0 = \frac{((\mathbf{p}-\mathbf{q})\times\mathbf{r})_z}{(\mathbf{s}\times\mathbf{r})_z} = \frac{((\mathbf{q}-\mathbf{p})\times\mathbf{r})_z}{(\mathbf{r}\times\mathbf{s})_z}. \tag{E39}$$

If $u_0, t_0 \in [0,1]$, one may conclude the line segments intersect. Otherwise, they do not.

Appendix F Numerical computation of the internal force density and macroelastic energy density

We present a numerical computation of the internal force density, $\mathbf{F}$, and macroelastic energy density, W, to check the correctness of their computation by the `ForceEnergyDensity` function (see Section 3.5). For this purpose, we consider a domain $\Omega = (0,1) \times (0,1)$ and we impose a deformation on all nodes in the domain. We choose a PD horizon $\delta = 0.2$ and a Young's modulus $E = 1$ in consistent units. To check the computation of W, we consider an isotropic extension with constant stretch $\bar{s} = 0.1$ for which $W = 8E\bar{s}^2/5$ for plane strain (see (C14)) and $W = 3E\bar{s}^2/2$ for plane stress (see (C17)). To check the computation of $\mathbf{F}$, we consider a quadratic deformation given by $\mathbf{u}(x,y) = (V_{11}x^2, 0)$ with $V_{11} = 0.01$ for which $\mathbf{F} \approx (12V_{11}E/5, 0)$ for plane strain and $\mathbf{F} \approx (9V_{11}E/4, 0)$ for plane stress (see (D26)).[6] Table F2 reports the corresponding results for a point in the bulk of the body, for a discretization given by a uniform grid of grid spacing $\Delta x = \Delta y = \delta/6$, using the function `TestForceEnergyDensity` based on the computations for uniform grids over rectangular domains (see Section 3.5); the same results are obtained based on the computations for general grids.

Remark 13 The results reported in Table F2 depend on the particular discretization chosen. A comparative study between the different partial-area algorithms is presented in [42]. Note that for the case of $\omega_{0.5}$, the accuracy of the PA-AC algorithm is reduced; this is expected since this choice of influence function does not allow evaluation of quadrature points outside the neighborhood, as opposed to ω_0.

Remark 14 The selection of problems and quantities reported in Table F2 was based on having quantities that neither vanish nor change for the different influence functions, for brevity. For the isotropic extension case, the internal force density, $\mathbf{F}$, vanishes for points in the bulk of a body, so we only report the macroelastic energy density, W. For the quadratic deformation case, the y-component of the internal force density, F_2, vanishes (see (D26b)) while the macroelastic energy density, W, becomes a function of the influence function (see Table D1), so we only report the x-component of the internal force density, F_1.

Appendix G Numerical computation of the fracture energy

We present a numerical computation of the fracture energy to confirm the expressions of critical stretch reported in Section 3.4 (see Table 3) by computing the numerical analogue of (C19), as described below.

[6] The choice of value for V_{11} was to ensure a small deformation, because the derivations in Appendix D are for a linearized model.

	ω_0	$\omega_{0.5}$	ω_1	ω_3	ω_5	ω_7	Continuum
Isotropic Extension							
Plane strain							
W: FA	1.60e-02	1.60e-02	1.59e-02	1.60e-02	1.60e-02	1.60e-02	
W: PA-AC	1.61e-02	1.49e-02	1.58e-02	1.60e-02	1.60e-02	1.60e-02	1.60e-02
W: IPA-AC	1.62e-02	1.62e-02	1.62e-02	1.60e-02	1.60e-02	1.60e-02	
Plane stress							
W: FA	1.50e-02	1.50e-02	1.49e-02	1.50e-02	1.50e-02	1.50e-02	
W: PA-AC	1.51e-02	1.40e-02	1.48e-02	1.50e-02	1.50e-02	1.50e-02	1.50e-02
W: IPA-AC	1.52e-02	1.52e-02	1.52e-02	1.50e-02	1.50e-02	1.50e-02	
Quadratic Deformation							
Plane strain							
F_1: FA	2.38e-02	2.38e-02	2.40e-02	2.41e-02	2.41e-02	2.41e-02	
F_1: PA-AC	2.42e-02	2.22e-02	2.39e-02	2.41e-02	2.41e-02	2.41e-02	2.40e-02
F_1: IPA-AC	2.44e-02	2.44e-02	2.44e-02	2.42e-02	2.41e-02	2.41e-02	
Plane stress							
F_1: FA	2.23e-02	2.23e-02	2.25e-02	2.26e-02	2.26e-02	2.26e-02	
F_1: PA-AC	2.27e-02	2.08e-02	2.24e-02	2.26e-02	2.26e-02	2.26e-02	2.25e-02
F_1: IPA-AC	2.28e-02	2.28e-02	2.29e-02	2.27e-02	2.26e-02	2.26e-02	

Table F2: Computation of the macroelastic energy density, W, and the x-component of the internal force density, F_1, at a node in the bulk of a body for the case of an isotropic extension with constant stretch $\bar{s} = 0.1$ and a quadratic deformation given by $\mathbf{u}(x, y) = (V_{11}x^2, 0)$ with $V_{11} = 0.01$, respectively. The computation employs $\delta = 0.2$ and a uniform grid of grid spacing $\Delta x = \Delta y = \delta/6$. The rightmost column provides reference values computed with the analytical expressions of the corresponding quantities for the continuum case. We report results for both plane strain and plane stress assumptions for the different influence functions. The values reported are computed with the `TestForceEnergyDensity` function.

Let us consider a vertical line representing a surface at $x = 0$. We numerically approximate the energy per unit area resulting from all the interactions between points along a horizontal line to the left of the vertical line and their neighboring points on the right side of the vertical line, under the assumption that all the corresponding bonds are critically stretched. For this purpose, we discretize the horizontal line with a set of segments of length Δx, and we discretize the right side of the vertical line with a uniform grid of cells of area $\Delta x \Delta y$. In practice, we sum all pairwise potentials of bonds connecting the nodes centered at the line segments to their neighboring cells while weighting each pairwise potential by the product of the segment length and the neighboring area. An illustration of the system is presented in Figure G4. This calculation is inspired by the numerical computations of nonlocal tractions presented in [55].

Choose a horizon $\delta = 1$ and grid spacing $\Delta x = \Delta y = \delta/6$, and let a domain be $\Omega = (-1, 1) \times (-1 - \frac{\Delta y}{2}, 1 + \frac{\Delta y}{2})$, which is uniformly discretized based on the grid spacing. Let the point intersecting the horizontal and vertical lines be $\mathbf{x}_0 = (x_0, y_0)$. Define the set $\mathcal{S}^{\mathrm{L}}$ of nodes on the horizontal line on the left side of the vertical line within Ω (see green nodes in Figure G4):

$$\mathcal{S}^{\mathrm{L}} := \{(x_0 - \tfrac{\Delta x}{2}, y_0), \ldots, (x_0 - \delta + \tfrac{\Delta x}{2}, y_0)\}$$

and the set $\mathcal{S}^{\mathrm{R}}$ of nodes on the right side of the vertical line within Ω (see black nodes in Figure G4) as:

$$\mathcal{S}^{\mathrm{R}} := \{x_0 + \tfrac{\Delta x}{2}, \ldots, x_0 + \delta - \tfrac{\Delta x}{2}, \} \times \{y_0 - \delta, \ldots, y_0 + \delta\},$$

which is built as a Cartesian product. Let $\mathbf{x}_i^{\mathrm{L}} \in \mathcal{S}^{\mathrm{L}}, i = 1, \ldots, N^{\mathrm{L}}$, and $\mathbf{x}_j^{\mathrm{R}} \in \mathcal{S}^{\mathrm{R}}, j = 1, \ldots, N^{\mathrm{R}}$, be the reference positions of the nodes within each set. The numerical analogue of (C19), G_0^h, can be expressed as follows:

$$G_0^h := h \sum_{i=1}^{N^{\mathrm{L}}} \sum_{j \in \hat{\mathcal{F}}_i^{\mathrm{R}}} \frac{1}{2} c\, \omega(\|\hat{\mathbf{x}}_{j(i)}^{\mathrm{R}} - \mathbf{x}_i^{\mathrm{L}}\|) s_0^2 \|\hat{\mathbf{x}}_{j(i)}^{\mathrm{R}} - \mathbf{x}_i^{\mathrm{L}}\| \Delta x \hat{A}_{j(i)}^{\mathrm{R}}, \tag{G40}$$

where $\hat{\mathcal{F}}_i^{\mathrm{R}}$ is a set of indices corresponding to the quadrature points for the two inner integrals in (C19), $\hat{\mathbf{x}}_{j(i)}^{\mathrm{R}}$ is the jth quadrature point, and $\hat{A}_{j(i)}^{\mathrm{R}}$ is the corresponding jth quadrature weight associated to node i located at $\mathbf{x}_i^{\mathrm{L}}$ (*cf.* (14)).

Let the Young's modulus be $E = 1$ and the fracture energy be $G_0 = 1$ in consistent units. We compute G_0^h for both plane strain and plane stress assumptions for the different influence functions, and we compare the results with G_0. These results are reported in Table G3. Note that the value of chs_0^2 is the same for both plane strain and plane stress, for each influence function, so we report both results together.

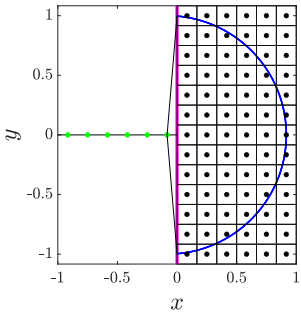


Fig. G4: Illustration of a domain discretization for the numerical computation of the fracture energy in (G40). We consider a vertical line representing a surface (thick magenta line) at $x = 0$. The computation considers all bonds connecting nodes on a horizontal line to the left of the vertical line (green nodes) and their neighbor cells on the right side of the vertical line. The intersecting point between the horizontal and vertical lines is $(x_0, y_0) = (0, 0)$. For clarity, the intersection between the boundary of the neighborhood of the rightmost node on the horizontal line and the cells to the right side of the vertical line is plotted in blue. The figure is produced with the `PlotNumericalFractureEnergy` function. Figure adapted from [55].

Plane strain / Plane stress							
	ω_0	$\omega_{0.5}$	ω_1	ω_3	ω_5	ω_7	G_0
G_0^h: FA	0.996	0.996	0.989	0.991	0.989	0.988	
G_0^h: PA-AC	1.006	0.913	0.980	0.990	0.989	0.988	1.000
G_0^h: IPA-AC	0.996	0.996	1.013	0.998	0.991	0.989	

Table G3: Computation of the fracture energy using (G40) for the system illustrated in Figure G4. The computation employs a horizon $\delta = 1$ and a uniform grid of grid spacing $\Delta x = \Delta y = \delta/6$. The Young's modulus is taken as $E = 1$ and the fracture energy is also taken as $G_0 = 1$ in consistent units. The micromodulus constant, c, and the critical stretch, s_0, are computed with the `PDBondConstants` function using the expressions in Section 3.4 (see Tables 2 and 3, respectively). The computations reported in the table are computed with the `TestCriticalStretch` function.

Appendix H Input decks for numerical examples

H.1 `WavePropagation` input deck

```
% ========================================================================
% Input deck for a wave propagation problem with an initial displacement
% based on a radial Gaussian distribution (the parameters are provided
% in consistent units)
% ========================================================================

% ------------------------------------------------------------------------
%                   Domain geometry and discretization
% ------------------------------------------------------------------------

% Domain boundaries
Xo = 0; % Left  boundary of the domain
Xn = 5; % Right boundary of the domain
Yo = 0; % Lower boundary of the domain
Yn = 5; % Upper boundary of the domain

% Number of nodes
Nx = 200; % Number of nodes in the x-direction
Ny = 200; % Number of nodes in the y-direction

% Grid perturbation coefficient
PG = 0;

% ------------------------------------------------------------------------
%                           Time discretization
% ------------------------------------------------------------------------

% Initial time
Ti = 0;

% Final time
Tf = 1.8;

% Time step
dt = 0.01;

% Time-integration scheme
TimeScheme = 'VVerlet'; % Velocity Verlet

% ------------------------------------------------------------------------
%                                PD model
% ------------------------------------------------------------------------

% Constitutive model
model = 'GPMB'; % Generalized Prototype Microelastic Brittle (GPMB) model

% Plane elasticity model
PlanarModel = 'PlaneStress';

% Horizon
del = 0.1;

% Influence function order indicator
omega = 0; % Constant influence function
```

```
% Flag for bond breaking
flag_BB = 0;

% ------------------------------------------------------------------------
%                       Classical material properties
% ------------------------------------------------------------------------

% Mass density
rho = 1;

% Young's modulus
E = 1;

% Fracture energy
Go = 1;

% ------------------------------------------------------------------------
%                          Meshfree discretization
% ------------------------------------------------------------------------

% Algorithm for computation of neighbor areas
AlgName = 'FA'; % FA algorithm

% ------------------------------------------------------------------------
%                            Problem settings
% ------------------------------------------------------------------------

% -------------------
% Body force density
% -------------------

bvfunc = @(x,y,t) (0.*x + 0.*y)*t; % x-component of body force density
bwfunc = @(x,y,t) (0.*x + 0.*y)*t; % y-component of body force density

% -------------------
% Initial conditions
% -------------------

% Initial displacement functions

% Tolerance
tol = 1E-15;

% Parameters
xm    = 2.5;     % x-coordinate of pulse center
ym    = 2.5;     % y-coordinate of pulse center
A     = 1;       % amplitude of radial Gaussian distribution
sigma = 1/30;    % standard deviation of radial Gaussian distribution
mu    = 6*sigma; % radial distance from pulse center of radial Gaussian
                 % distribution mean

% Functions
r      = @(x,y) sqrt((x-xm).^2+(y-ym).^2);       % distance from pulse center
uo     = @(x,y) ( r(x,y) > mu-6*sigma & r(x,y) < mu+6*sigma ).* A
           .* exp( (-(r(x,y) - mu).^2) / (2*sigma)^2 ); % magnitude of initial
                                                        % displacement
vofunc = @(x,y) uo(x,y).*(x-xm)./(r(x,y) + tol); % x-component of initial
                                                 % displacement
wofunc = @(x,y) uo(x,y).*(y-ym)./(r(x,y) + tol); % y-component of initial
                                                 % displacement
```

```
% Initial velocity functions

Vvofunc = @(x,y) 0.*x + 0.*y; % x-component of initial velocity
Vwofunc = @(x,y) 0.*x + 0.*y; % y-component of initial velocity

% ------------------------------------------------------------------------
%                            Postprocessing
% ------------------------------------------------------------------------

% Flag for plotting during time integration
flag_DynamicPlotting = 1;

% Frequency of plotting during time integration
DynamicPlotFrequency = 10; % Plot every 10 time steps
                           % (beginning from the first one)

% Frequency of time-integration step display
TimeStepDisplayFrequency = 10;

% Flag for plotting at final time
flag_FinalPlots = 1;

% Plot settings
%                     Field Name           Field variable    Colorbar title
PlotSettings = {'DisplacementMagnitude','sqrt(v.^2 + w.^2)','$\| bf u \|$',
                    Point size  Colormap limits  Colormap    Axes limits
                          8    ,    [0 1.0]    , 'parula' , [Xo Xn Yo Yn] ,
                    Configuration
                     'Reference'};

% ------------------------------------------------------------------------
```

H.2 CrackBranching input deck

```
% ========================================================================
% Input deck for a crack branching problem on a pre-notched soda-lime
% glass thin plate
% ========================================================================

% ------------------------------------------------------------------------
%                  Domain geometry and discretization
% ------------------------------------------------------------------------

% Domain boundaries
Xo = -0.05; % [m]:  Left  boundary of the domain
Xn =  0.05; % [m]:  Right boundary of the domain
Yo = -0.02; % [m]:  Lower boundary of the domain
Yn =  0.02; % [m]:  Upper boundary of the domain

% Number of nodes
Nx = 300; % Number of nodes in the x-direction
Ny = 120; % Number of nodes in the y-direction

% Grid perturbation coefficient
PG = 0;
```

```
% ------------------------------------------------------------------------
%                            Time discretization
% ------------------------------------------------------------------------

% Initial time
Ti = 0;

% Final time
Tf = 4.3e-5; % [s] :    43 microsec

% Time step
dt = 6.7E-8; % [s] : 67E-3 microsec

% Time-integration scheme
TimeScheme = 'VVerlet'; % Velocity Verlet

% ------------------------------------------------------------------------
%                                 PD model
% ------------------------------------------------------------------------

% Constitutive model
model = 'GPMB'; % Generalized Prototype Microelastic Brittle (GPMB) model

% Plane elasticity model
PlanarModel = 'PlaneStress';

% Horizon
del = 0.001; % [m]

% Influence function order indicator
omega = 0; % Constant influence function

% Flag for bond breaking
flag_BB = 1;


% ------------------------------------------------------------------------
%                       Classical material properties
% ------------------------------------------------------------------------

% Mass density
rho = 2440; % [kg/m^3]

% Young's modulus
E = 72e+9;  % [Pa] : 72 GPa

% Fracture energy
Go = 3.8;   % [J/m^2]

% ------------------------------------------------------------------------
%                           Meshfree discretization
% ------------------------------------------------------------------------

% Algorithm for computation of neighbor areas
AlgName = 'FA'; % FA algorithm
```

```
% ------------------------------------------------------------------------
%                            Problem settings
% ------------------------------------------------------------------------

% ------------------
% Body force density
% ------------------

% Compute dy
dy = (Yn - Yo)/Ny;

% Traction amplitude
sigma = 2E6; % [Pa] : 2 MPa

bvfunc = @(x,y,t) (0.*x + 0.*y)*t;                          % x-component of
                                                            % body force density
bwfunc = @(x,y,t) (abs(y) > Yn - dy).*sigma.*sign(y)/dy; % y-component of
                                                            % body force density

% ------------------
% Initial conditions
% ------------------

% Initial displacement functions

vofunc = @(x,y) 0.*x + 0.*y; % x-component of initial displacement
wofunc = @(x,y) 0.*x + 0.*y; % y-component of initial displacement

% Initial velocity functions

Vvofunc = @(x,y) 0.*x + 0.*y; % x-component of initial velocity
Vwofunc = @(x,y) 0.*x + 0.*y; % y-component of initial velocity

% --------------------
% No-fail zone
% --------------------

% No-fail function
nofailfunc = @(x,y) ( abs(y) > Yn - del );

% ------------------
% Prenotch
% ------------------

%                        Xc1   Yc1  Xc2  Yc2
PreNotchCoordinates = [-0.05  0.0  0.0  0.0];

% ------------------------------------------------------------------------
%                             Postprocessing
% ------------------------------------------------------------------------

% Flag for plotting during time integration
flag_DynamicPlotting = 1;

% Frequency of plotting during time integration
DynamicPlotFrequency = 40; % Plot every 40 time steps
                           % (beginning from the first one)

% Frequency of time-integration step display
TimeStepDisplayFrequency = 20;
```

```
% Flag for plotting at final time
flag_FinalPlots = 1;

% Plot settings
%                      Field Name           Field variable    Colorbar title
PlotSettings = {'StrainEnergyDensity'  ,    'log10(W)'    , '$\log_10(W)$',
                    Point size  Colormap limits  Colormap    Axes limits
                          8     ,    [0 3.5]    ,  'jet'    , [Xo Xn Yo Yn] ,
                    Configuration
                     'Reference';
                      Field Name           Field variable    Colorbar title
                       'Damage'          ,       'phi'      ,  '$\varphi$'  ,
                    Point size  Colormap limits  Colormap    Axes limits
                          8     ,    [0 0.4]    , 'parula' , [Xo Xn Yo Yn] ,
                    Configuration
                     'Reference' };

% Flag to visualize pre-notch as damaged
flag_DamagedPrenotches = 1;

% ------------------------------------------------------------------------
```

Declarations

Ethical Approval. Not applicable

Competing interests. The authors have no competing interests to declare that are relevant to the content of this article.

Authors' contributions. P.S. led the overall code development and paper writeup, supervised Y.J. on the writing of the initial version of the code and numerical examples, worked with M.P. on updating the initial version of the code, wrote most of the sections of the paper, and performed the analytical derivations. M.P. assisted in updating the initial version of the code as well as on the writeup of many sections of the paper, and he wrote the initial version of the numerical examples' sections. Y.J. wrote the initial version of the code and numerical examples. J.T. wrote the appendix of the paper on the intersection of two line segments and an initial version of some visualization functions, and he optimized and cleaned up several parts of the code. S.T.R wrote the literature review of peridynamics codes and handled the software version control and release. All authors reviewed the manuscript.

Funding. Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC for the US Department of Energy under contract DE-AC05-00OR22725. This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship program.

Availability of data and materials. The code described in this work is openly available: https://github.com/ORNL/PDMATLAB2D, including

direct reproduction of the numerical examples and demonstrations within the paper.

References

[1] Silling, S.A.: Reformulation of elasticity theory for discontinuities and long-range forces. Journal of the Mechanics and Physics of Solids **48**(1), 175–209 (2000). https://doi.org/10.1016/S0022-5096(99)00029-0

[2] Dahal, B., Seleson, P., Trageser, J.: The evolution of the peridynamics co-authorship network. Journal of Peridynamics and Nonlocal Modeling (2022). https://doi.org/10.1007/s42102-022-00082-5

[3] Parks, M.L., Lehoucq, R.B., Plimpton, S.J., Silling, S.A.: Implementing peridynamics within a molecular dynamics code. Computer Physics Communications **179**(11), 777–783 (2008). https://doi.org/10.1016/j.cpc.2008.06.011

[4] Parks, M.L., Seleson, P., Plimpton, S.J., Silling, S.A., Lehoucq, R.B.: Peridynamics with LAMMPS: A user guide v0.3 beta. Technical Report SAND2011-8523, Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550 (November 2011). https://doi.org/10.2172/1031301

[5] Thompson, A.P., Aktulga, H.M., Berger, R., Bolintineanu, D.S., Brown, W.M., Crozier, P.S., in 't Veld, P.J., Kohlmeyer, A., Moore, S.G., Nguyen, T.D., Shan, R., Stevens, M.J., Tranchida, J., Trott, C., Plimpton, S.J.: LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Computer Physics Communications **271**, 108171 (2022). https://doi.org/10.1016/j.cpc.2021.108171

[6] Plimpton, S.J., Kohlmeyer, A., Thompson, A.P., Moore, S.G., Berger, R.: LAMMPS: Large-scale Atomic/Molecular Massively Parallel Simulator. https://doi.org/10.5281/zenodo.3726416

[7] Parks, M.L., Littlewood, D.J., Mitchell, J.A., Silling, S.A.: Peridigm users' guide v1.0.0. Technical Report SAND2012-7800, Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550 (September 2012). https://doi.org/10.2172/1055619

[8] Peridigm. https://github.com/peridigm/peridigm

[9] Heroux, M.A., Willenbring, J.M.: A new overview of the Trilinos project. Scientific Programming **20**(2), 83–88 (2012). https://doi.org/10.3233/SPR-2012-0355

[10] Jafarzadeh, S., Mousavi, F., Bobaru, F.: PeriFast/Dynamics: a MATLAB code for explicit fast convolution-based peridynamic analysis of deformation and fracture (2022). https://doi.org/10.21203/rs.3.rs-2019917/v1

[11] Wang, L., Jafarzadeh, S., Bobaru, F.: PeriFast/Corrosion: a 3D pseudo-spectral peridynamic Matlab code for corrosion (2022). https://doi.org/10.21203/rs.3.rs-2046856/v1

[12] PeriFast. https://github.com/PeriFast/Code

[13] Han, F., Li, Z.: A peridynamics-based finite element method (PeriFEM) for quasi-static fracture analysis. Acta Mechanica Solida Sinica **35**(3), 446–460 (2022). https://doi.org/10.1007/s10338-021-00307-y

[14] Jha, P.K., Desai, P.S., Bhattacharya, D., Lipton, R.: Peridynamics-based discrete element method (PeriDEM) model of granular systems involving breakage of arbitrarily shaped particles. Journal of the Mechanics and Physics of Solids **151**, 104376 (2021). https://doi.org/10.1016/j.jmps.2021.104376

[15] Jha, P.K.: PeriDEM. https://doi.org/10.5281/zenodo.4733259

[16] Li, X., Ye, H., Zhang, J.: Large-scale simulations of peridynamics on Sunway Taihulight supercomputer. In: 49th International Conference on Parallel Processing-ICPP, pp. 1–11 (2020). https://doi.org/10.1145/3404397.3404421

[17] Wang, X., Wang, Q., An, B., He, Q., Wang, P., Wu, J.: A GPU parallel scheme for accelerating 2D and 3D peridynamics models. Theoretical and Applied Fracture Mechanics **121**, 103458 (2022). https://doi.org/10.1016/j.tafmec.2022.103458

[18] Zhong, J., Han, F., Zhang, L.: Accelerated peridynamic computation on GPU for quasi-static fracture simulations (2022). https://doi.org/10.21203/rs.3.rs-1937120/v1

[19] Mossaiby, F., Shojaei, A., Zaccariotto, M., Galvanetto, U.: OpenCL implementation of a high performance 3D Peridynamic model on graphics accelerators. Computers & Mathematics with Applications **74**(8), 1856–1870 (2017). https://doi.org/10.1016/j.camwa.2017.06.045

[20] Boys, B., Dodwell, T.J., Hobbs, M., Girolami, M.: PeriPy-A high performance OpenCL peridynamics package. Computer Methods in Applied Mechanics and Engineering **386**, 114085 (2021). https://doi.org/10.1016/j.cma.2021.114085

[21] PeriPy. https://github.com/alan-turing-institute/PeriPy

[22] Jha, P.K., Diehl, P.: NLMech: Implementation of finite difference/meshfree discretization of nonlocal fracture models. Journal of Open Source Software **6**(65), 3020 (2021). https://doi.org/10.21105/joss.03020

[23] Diehl, P., Jha, P.K.: NLMech: Release for the JOSS Paper. https://doi.org/10.5281/zenodo.5532697

[24] Reeve, S., Seleson, P.: CabanaPD. https://doi.org/10.5281/zenodo.7087781

[25] Dark, J., Sansom, K., Littlewood, D., Trageser, J., Wolf, I., Patton, C.: Peridot (2019). https://doi.org/10.11578/dc.20191118.1

[26] Silling, S.A., Askari, E.: A meshfree method based on the peridynamic model of solid mechanics. Computers & Structures **83**(17–18), 1526–1535 (2005). https://doi.org/10.1016/j.compstruc.2004.11.026

[27] Emmrich, E., Weckner, O.: The peridynamic equation and its spatial discretisation. Mathematical Modelling and Analysis **12**(1), 17–27 (2007). https://doi.org/10.3846/1392-6292.2007.12.17-27

[28] Chen, X., Gunzburger, M.: Continuous and discontinuous finite element methods for a peridynamics model of mechanics. Computer Methods in Applied Mechanics and Engineering **200**(9–12), 1237–1250 (2011). https://doi.org/10.1016/j.cma.2010.10.014

[29] Ren, B., Wu, C.T., Askari, E.: A 3D discontinuous Galerkin finite element method with the bond-based peridynamics model for dynamic brittle failure analysis. International Journal of Impact Engineering **99**, 14–25 (2017). https://doi.org/10.1016/j.ijimpeng.2016.09.003

[30] Pasetto, M., Leng, Y., Chen, J.-S., Foster, J.T., Seleson, P.: A reproducing kernel enhanced approach for peridynamic solutions. Computer Methods in Applied Mechanics and Engineering **340**, 1044–1078 (2018). https://doi.org/10.1016/j.cma.2018.05.010

[31] Trask, N., You, H., Yu, Y., Parks, M.L.: An asymptotically compatible meshfree quadrature rule for nonlocal problems with applications to peridynamics. Computer Methods in Applied Mechanics and Engineering **343**, 151–165 (2019). https://doi.org/10.1016/j.cma.2018.08.016

[32] Shojaei, A., Hermann, A., Cyron, C.J., Seleson, P., Silling, S.A.: A hybrid meshfree discretization to improve the numerical performance of peridynamic models. Computer Methods in Applied Mechanics and Engineering **391**, 114544 (2022). https://doi.org/10.1016/j.cma.2021.114544

[33] Seleson, P., Pasetto, M., John, Y., Trageser, J.: PDMATLAB2D: Version 1.0. https://doi.org/10.5281/zenodo.7348668

[34] Bobaru, F., Foster, J.T., Geubelle, P.H., Silling, S.A. (eds.): Handbook of Peridynamic Modeling, 1st edn. Chapman and Hall/CRC, New York, NY (2016). https://doi.org/10.1201/9781315373331

[35] Silling, S.A., Epton, M., Weckner, O., Xu, J., Askari, E.: Peridynamic states and constitutive modeling. Journal of Elasticity **88**(2), 151–184 (2007). https://doi.org/10.1007/s10659-007-9125-1

[36] Silling, S.A., Lehoucq, R.B.: Peridynamic theory of solid mechanics. In: Aref, H., van der Giessen, E. (eds.) Advances in Applied Mechanics. Advances in Applied Mechanics, vol. 44, pp. 73–168. Elsevier, Amsterdam, The Netherlands (2010). https://doi.org/10.1016/S0065-2156(10)44002-8

[37] Madenci, E., Oterkus, E.: Peridynamic Theory and Its Applications, 1st edn. Springer, New York, NY (2014). https://doi.org/10.1007/978-1-4614-8465-3

[38] Gerstle, W., Sau, N., Silling, S.: Peridynamic modeling of plain and reinforced concrete structures. In: 18th International Conference on Structural Mechanics in Reactor Technology (SMiRT 18), pp. 54–68 (2005). http://www.lib.ncsu.edu/resolver/1840.20/31420

[39] Seleson, P., Parks, M.: On the role of the influence function in the peridynamic theory. International Journal for Multiscale Computational Engineering **9**(6), 689–706 (2011). https://doi.org/10.1615/IntJMultCompEng.2011002527

[40] Seleson, P.D.: Peridynamic multiscale models for the mechanics of materials: constitutive relations, upscaling from atomistic systems, and interface problems. PhD thesis, The Florida State University (2010). http://purl.flvc.org/fsu/fd/FSU_migr_etd-0273

[41] Foster, J.T., Silling, S.A., Chen, W.: An energy based failure criterion for use with peridynamic states. International Journal for Multiscale Computational Engineering **9**(6), 675–688 (2011). https://doi.org/10.1615/IntJMultCompEng.2011002407

[42] Seleson, P.: Improved one-point quadrature algorithms for two-dimensional peridynamic models based on analytical calculations. Computer Methods in Applied Mechanics and Engineering **282**, 184–217 (2014). https://doi.org/10.1016/j.cma.2014.06.016

[43] Seleson, P., Littlewood, D.J.: Convergence studies in meshfree peridynamic simulations. Computers & Mathematics with Applications **71**(11), 2432–2448 (2016). https://doi.org/10.1016/j.camwa.2015.12.021

[44] Seleson, P., Littlewood, D.J.: In: Voyiadjis, G.Z. (ed.) Numerical Tools for Improved Convergence of Meshfree Peridynamic Discretizations, pp. 1–27. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-22977-5_39-1

[45] Bobaru, F., Ha, Y.D.: Adaptive refinement and multiscale modeling in 2D peridynamics. International Journal for Multiscale Computational Engineering **9**(6), 635–660 (2011). https://doi.org/10.1615/IntJMultCompEng.2011002793

[46] Yu, K., Xin, X.J., Lease, K.B.: A new adaptive integration method for the peridynamic theory. Modelling and Simulation in Materials Science and Engineering **19**(4), 045003 (2011). https://doi.org/10.1088/0965-0393/19/4/045003

[47] Zheng, G., Wang, J., Shen, G., Xia, Y., Li, W.: A new quadrature algorithm consisting of volume and integral domain corrections for two-dimensional peridynamic models. International Journal of Fracture **229**(1), 39–54 (2021). https://doi.org/10.1007/s10704-021-00540-z

[48] Scabbia, F., Zaccariotto, M., Galvanetto, U.: Accurate computation of partial volumes in 3D peridynamics. Engineering with Computers (2022). https://doi.org/10.1007/s00366-022-01725-3

[49] Hairer, E., Lubich, C., Wanner, G.: Geometric numerical integration illustrated by the Störmer–Verlet method. Acta Numerica **12**, 399–450 (2003). https://doi.org/10.1017/S0962492902000144

[50] Seleson, P., Parks, M.L., Gunzburger, M., Lehoucq, R.B.: Peridynamics as an upscaling of molecular dynamics. Multiscale Modeling & Simulation **8**(1), 204–227 (2009). https://doi.org/10.1137/09074807X

[51] Ha, Y.D., Bobaru, F.: Studies of dynamic crack propagation and crack branching with peridynamics. International Journal of Fracture **162**(1-2), 229–244 (2010). https://doi.org/10.1007/s10704-010-9442-4

[52] F. Bobaru, G. Zhang: Why do cracks branch? A peridynamic investigation of dynamic brittle fracture. International Journal of Fracture **196**, 59–98 (2015). https://doi.org/10.1007/s10704-015-0056-8

[53] Trageser, J., Seleson, P.: Bond-based peridynamics: A tale of two Poisson's ratios. Journal of Peridynamics and Nonlocal Modeling **2**(3), 278–288 (2020). https://doi.org/10.1007/s42102-019-00021-x

[54] Goldman, R.: Intersection of two lines in three-space. In: Glassner, A.S. (ed.) Graphics Gems, p. 304. Morgan Kaufmann, San Diego. CA (1990). https://doi.org/10.1016/B978-0-08-050753-8.50064-4

[55] Ongaro, G., Seleson, P., Galvanetto, U., Ni, T., Zaccariotto, M.: Overall equilibrium in the coupling of peridynamics and classical continuum mechanics. Computer Methods in Applied Mechanics and Engineering **381**, 113515 (2021). https://doi.org/10.1016/j.cma.2020.113515