Journal of Peridynamics and Nonlocal Modeling
https://doi.org/10.1007/s42102-023-00097-6

RESEARCH

PeriFast/Dynamics: A MATLAB Code for Explicit Fast Convolution-based Peridynamic Analysis of Deformation and Fracture

Siavash Jafarzadeh[1,2] · Farzaneh Mousavi[1] · Longzhen Wang[1] · Florin Bobaru[1]

Received: 1 September 2022 / Accepted: 8 March 2023

Abstract
We present PeriFast/Dynamics, a compact and user-friendly MATLAB code for fast peridynamic (PD) simulations for deformation and fracture. PeriFast/Dynamics uses the fast convolution-based method (FCBM) for spatial discretization and an explicit time marching scheme to solve large-scale dynamic fracture problems. Different from existing PD solvers, PeriFast/Dynamics does not require neighbor search and storage, due to the use of the Fast-Fourier transform and its inverse to compute the integral operator. Run-times and memory allocation are independent of the number of neighbors inside the PD horizon, leading to faster computations and lower storage requirements. The governing equations and discretization method are briefly reviewed, the code structure explained, and individual modules described in detail. A demonstrative example on dynamic brittle fracture in 3D, with multiple crack branching events, is solved using three different constitutive models: a bond-based, an ordinary state-based, and a correspondence model. The small differences between results with the three different constitutive models are explained. Users are provided with a step-by-step description of the problem setup and execution of the code. PeriFast/Dynamics is a branch of the PeriFast suite of codes, and is available for download at the GitHub link provided in reference [1].

Keywords Peridynamics · Software · MATLAB · Fast Fourier transform · Dynamic fracture · Convolution

Siavash Jafarzadeh and Farzaneh Mousavi are co-first authors and equally contributed to this work.

✉ Florin Bobaru
fbobaru2@unl.edu

[1] Department of Mechanical and Materials Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0526, USA

[2] Department of Civil and Environmental Engineering, The Pennsylvania State University, University Park, PA 16802-1408, USA

1 Introduction

Computational modeling of damage and fracture has been one of the most challenging areas in computational mechanics. Classical theories with the governing equations expressed in terms of partial differential equations (PDEs) are not fully capable of describing fracture since cracks are, in fact, evolving discontinuities in the continuum, and spatial derivatives at discontinuities in the displacement field are not defined. Peridynamic formulations for mechanics [2] offer alternative nonlocal approaches in which spatial derivatives are replaced with volume integrals of the primary unknowns over a certain finite region around each point, hence, allowing discontinuities (in the unknown field) to emerge and evolve in a mathematical consistent way since integration is not affected by discontinuities. PD makes seamless modeling of fracture and damage possible. In PD, cracks can naturally emerge, propagate, branch, and coalesce without the need of external, ad-hoc rules and conditions (e.g., see [3–5]). Significant interest on modeling fracture with PD has been observed [6–8].

The most common, straightforward and functional discretization for PD equations is the so-called meshfree method. In this, one approximates the integral over the nonlocal region (the PD horizon region) with a Riemann-type sum, normally using the one-point Gaussian integration, or a slight modification of that to account for nodal volumes (usually cubes) that are only partially covered by the PD horizon region [9]. Note that the commercially available computer-aided engineering (CAE) software is mostly based on the finite element method (FEM) and classical PDEs. Consequently, they are inherently different from meshfree PD in terms of data structures for geometry (elements and quadrature nodes in FEM, versus nodes in meshfree PD), and in terms of solvers used, since they are based on different numerical approximation methods. There have been several attempts to manipulate commercial FEM packages to perform PD analyses (e.g., see [10, 11]). Some commercial codes, e.g., LS-Dyna, have added PD capabilities as separate modules in their platform. In LS-Dyna, for example, the Discontinuous Galerkin method is used to approximate solutions to PD models ([12, 13]). U.S. National labs like Sandia and Oak Ridge National Laboratories, and research groups in academia and research labs in industry developed in-house codes for PD. Peridigm [14] is one of the few open-source PD software available from Sandia. The MOOSE-based PD code for implicit thermomechanical analysis by Idaho National Laboratory [15] is another example.

Because of its versatility in solving problems in fracture and damage, the meshfree method with direct summation for the quadrature is adopted by most existing PD in-house codes. In this approach, at every node, a loop is performed over all nodes in its "family" (neighboring nodes positioned within a finite size distance from the current node). If N is the total number of nodes and M is the number of nodes in the family of an arbitrary node, the nested loops result in solvers with the computational complexity of, at best, $O(NM)$. In 3D PD simulations with coarsest grids, M is at least in order of hundreds, which make PD simulations costly when compared with, for example, FEM solvers for corresponding local models. Using FEM solvers for PD is, obviously, an option but the complexity would be the same; in addition, FEM solvers are not practical for solving problems with discontinuities. That is where the advantage of the meshfree method comes in. These observations show the need for faster solvers for PD models, especially for problems involving discontinuities, like fracture and damage.

Various attempts have been made to reduce the cost of PD simulations. One popular approach is the local-nonlocal coupling where only areas around cracks are modeled by

PD, while the rest of the body is modeled using the local theory, discretized by FEM or by an efficient meshfree method [16–18]. Another way pursued for addressing PD high computational cost is grid refinement where necessary (mostly areas with damage) [19, 20]. These approaches, however, require prior knowledge of where fracture is likely to occur, or they need to employ smart adaptive schemes for determining those zones, which adds challenges and coding complexities. Moreover, for cases in which the damaged and fragmented region comprises most of the body, these coupling/adaptive approaches lose their advantage relative to a full PD model.

Recently, the fast convolution-based method (FCBM) for PD was introduced [21–23]. In this method, nodal quadrature is expressed in terms of convolution sums, which are evaluated efficiently via fast Fourier transform operations. Since the quadrature is evaluated by multiplication of the Fourier modes of the convolving functions, looping over the family of neighbors is not performed. The major cost is associated with the FFT operations, which leads to an $O\left(N\log_2 N\right)$ complexity. No neighbor search and storage are required in FCBM, resulting in fast initialization and low memory requirements. The studies mentioned above have shown speedups as high as thousands compared with the direct quadrature PD solver. Certain variations of FCBM have been proposed and used in [24–27].

In the original meshfree method with direct summation quadrature, for a fixed horizon size, M scales with $O(N)$ in an m-convergence test [28], leading to $O\left(N^2\right)$ complexity. For a δ-convergence calculation [28], in which M is kept fixed while reducing the horizon size (thus increasing N), the direct summation quadrature would scale as $O(NM)$ which, at first glance, looks better than $O\left(N\log_2 N\right)$. However, based on the performance comparison between direct quadrature and FCBM, for an m-factor (ratio of horizon size to grid spacing) of 3 in a 3D computation, the direct summation quadrature outperforms FCBM only when N becomes larger than 7×10^{58}.

Another advantage of the FCBM-PD is its low barrier for utilizing HPC. One can simply call parallel or GPU-based FFT libraries instead of the serial ones to benefit from parallel computing at no additional programming effort. An FCBM for PD diffusion code has been published as supplementary material in a previous study [22]. That code, however, was designed for a limited class of diffusion problems.

Here, we introduce PeriFast/Dynamics, an FCBM-based code written in MATLAB for modeling deformation and fracture using peridynamics. PeriFast/Dynamics is simple, compact, and easy to use and expand to a variety of other problems. After brief reviews of PD formulations for deformations/fracture/damage and the main steps in the FCBM for discretizing such formulations, we provide the overall description of the PeriFast/Dynamics code structure with detailed explanations for each of its modules (m-files). In addition, we present an example showing how the code can be used, and how one can extend it to other material models and applications. Full details of the PD theory and FCBM discretization can be found in the following references, for example: [22, 23, 29, 30].

The paper is organized as follows: governing equations for the initial boundary value problems for deformations and damage are discussed in Section 2; in Section 3, the FCBM discretization is briefly reviewed; data structures used in PeriFast/Dynamics are shown in Section 4; the code's general structure and details of each of its modules are presented in Section 5; in Section 6, a demonstrative example is provided with step-by-step instructions for a user to perform a 3D peridynamic analysis of dynamic brittle fracture using the freely-available code.

2 The Peridynamic Initial-Value Volume-Constrained Problem for Dynamic Fracture

PeriFast/Dynamics analysis aims to solve PD equations of dynamic deformation and damage subjected to initial conditions (IC) and volume constraints (VC), a.k.a. nonlocal boundary conditions. Consider a 3D peridynamic body (B), with constrained volumes Γ_1, Γ_2, and Γ_3 on which the displacement field components u_1, u_2, and u_3 are respectively prescribed. The constrained volumes usually coincide with one another, but they do not have to. Figure 1 shows a generic 2D PD body with constrained volumes.

Let $\boldsymbol{x}(t) = \{x_1(t), x_2(t), x_3(t)\}$ be the position vector of a material point at time t, with $i = 1, 2, 3$ corresponding to the three Cartesian coordinate directions in 3D. The PD initial-value volume-constrained (IVVC) problem for dynamics is [31]

$$\begin{cases} \rho \frac{\partial^2 u_i(\boldsymbol{x},t)}{\partial t^2} = L_i(\boldsymbol{x}, t) + b_i(\boldsymbol{x}, t) & \boldsymbol{x} \in \Omega_i,\ t > 0 \\ u_i(\boldsymbol{x}, 0) = u_i^0;\ v_i(\boldsymbol{x}, 0) = v_i^0 (\text{I.C.}) & \boldsymbol{x} \in \Omega_i \\ u_i(\boldsymbol{x}, t) = g_i(\boldsymbol{x}, t) (\text{V.C.}) & \boldsymbol{x} \in \Gamma_i,\ t \geq 0 \end{cases} ;\ i = 1, 2, 3 \tag{1}$$

where u_i is the displacement in the i-direction, v_i(velocity) is the time-derivative of u_i, g_i is a given volume constraints on Γ_i, and b_i is the body/external force density in the i-direction. L_i denotes the internal force density in i-direction and is defined as

$$L_i(\boldsymbol{x}, t) = \int_{\mathcal{H}_x} \mu\left(\boldsymbol{x}, \boldsymbol{x}', t\right) f_i\left(\boldsymbol{x}, \boldsymbol{x}', t\right) \mathrm{d}\boldsymbol{x}';\ i = 1, 2, 3 \tag{2}$$

where $\mathcal{H}_x$ is the finite size neighborhood of $\boldsymbol{x}$ where the nonlocal interactions pertaining to $\boldsymbol{x}$ occur. $\mathcal{H}_x$ is known as the *family or the horizon region* of point $\boldsymbol{x}$ and is usually a sphere in 3D, centered at $\boldsymbol{x}$ with the radius δ referred to as the *horizon* size. $\boldsymbol{x}'$ denotes the position vector for family nodes in $\mathcal{H}_x$. $f_i\left(\boldsymbol{x}, \boldsymbol{x}', t\right)$ is the *dual force density*: the net force between the material volume at $\boldsymbol{x}$ and the material volume at $\boldsymbol{x}'$, and is determined by a PD constitutive model. PD material models, which define the expression for $f_i\left(\boldsymbol{x}, \boldsymbol{x}', t\right)$ in Eq. (2), can be of two types: bond-based (BB) and state-based (SB). In BB-PD, the dual force density for each pair of nodes depends on the displacement of those nodes only, whereas in the more general SB-PD, the dual force density for each pair of nodes can depend on the deformation of the entire families of $\boldsymbol{x}$ and $\boldsymbol{x}'$. In SB-PD, PD states are introduced as general nonlinear mappings, generalizations of tensors, which are linear mappings, in the

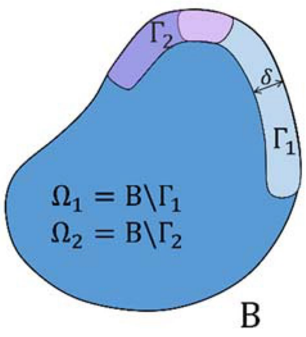

Γ_2
δ
Γ_1
$\Omega_1 = \mathrm{B}\backslash\Gamma_1$
$\Omega_2 = \mathrm{B}\backslash\Gamma_2$
B

Fig. 1 Schematic of a 2D peridynamic body (B), consisting of the domains Ω_1 and Ω_2, where displacement components u_1 and u_2 are unknown, respectively, and the constrained volumes (Γ_1 and Γ_2) where u_1 and u_2 are independently prescribed. (Figure adopted from [23])

classical continuum mechanics theory [32]. The constitutive relationships define the PD "force-state" as a function of the PD "deformation-state" and other quantities. f_i in Eq. (2) is defined based on the force-states at $\boldsymbol{x}$ and $\boldsymbol{x'}$ [32]. The relationship between PD force and deformation states can either be directly constructed/obtained in the nonlocal setting (the "native PD approach"), or it can be derived by a conversion (or "translation") method from a classical (local) constitutive model. The latter is known as the *PD correspondence approach*, which usually leads to non-ordinary state-based (NOSB) PD models. In ordinary state-based (OSB) PD models, the force vector between $\boldsymbol{x}$ and $\boldsymbol{x}'$ is collinear to the bond vector connecting the two points, while NOSB-PD models this does not necessarily happen [32]. Correspondence models are convenient since they can use existing constitutive local models, but can suffer from numerical instabilities (zero energy modes, see [33–35]), and tend to have a higher computational cost than corresponding OSB ones. The constitutive model formulas used in this work are given in Appendix.

The function μ in Eq. (2) is a history-dependent bond-level damage function with the following binary definition normally used for brittle-type damage:

$$\mu(\boldsymbol{x},\boldsymbol{x'},t) = \begin{cases} 1 & \boldsymbol{xx'} \text{ bond is intact (carries force) at time } t \\ 0 & \boldsymbol{xx'} \text{ bond is broken (does not carry a force) at time } t \end{cases} \tag{3}$$

Note that PD *bonds* refer to pairs of family points. A broken bond means that the interaction between the two family points that the bond connects no longer exists. In PeriFast/Dynamics, we use the energy-based damage model proposed in [23], which is consistent with the FCBM discretization. In this model, once the strain energy density (W) at a point reaches a critical strain energy density (W_c), that point loses all of its bonds irreversibly, i.e., it is completely detached from the body. The definition for μ in this approach can be expressed as

$$\mu\left(\boldsymbol{x},\boldsymbol{x}',t\right) = \lambda(\boldsymbol{x},t)\lambda\left(\boldsymbol{x}',t\right) = \lambda\lambda' \tag{4}$$

where

$$\lambda(\boldsymbol{x},t) = \begin{cases} 1 & \text{if } W(\boldsymbol{x},t) \le W_c \\ 0 & \text{if } W(\boldsymbol{x},t) > W_c \end{cases} \tag{5}$$

The definition of $W(\boldsymbol{x},t)$ depends on the constitutive model. For the material models implemented in PeriFast, W is provided in Appendix. The threshold W_c is calibrated to the critical fracture energy G_0 :

$$W_c = \frac{G_0}{2\delta} \tag{6}$$

The details of calibration can be found in [23]. Note that the calibrated formula shown in reference [23] does not include the 2 in the denominator. There, the 2 was incorporated into formula for W, leading to equivalent results as here. We prefer the current formula for clarity.

Most engineering measurements are taken on surfaces of the domain, leading to mathematical descriptions in terms of (classical) Dirichlet, Neumann, or mixed boundary conditions. In order to approximate a (classical) Dirichlet boundary condition in the PD nonlocal settings described by Eq. (1), one can impose displacements on a δ-thick volumetric layer at the boundary: this is known as the "naïve approach" [36]. For more accurate enforcement of local boundary conditions in PD models, please see, e.g., [36–40]. In the current version of PeriFast/Dynamics, we use the naïve approach. The mirror-based fictitious nodes method (FNM) [36] is also compatible with FCBM and has been implemented in the PeriFast/Corrosion branch [41].

Traction boundary conditions (Neumann type) are usually implemented as body force densities applied on a δ-thick layer at the corresponding boundary. Other options can be used, for example, one can specify a certain profile for g_i in Eq. (1), that approximates, for example, the desired Dirichlet and Neumann boundary conditions, see [22, 36]. The body force approach is implemented here in PeriFast/Dynamics.

In order to be able to use the FCBM-PD, a constitutive model needs to be setup in convolutional form. For the PeriFast/Dynamics code, the linearized BB, linearized native OSB-PD model, and PD correspondence model are implemented based on the formulations presented in [23, 42], where the convolutional forms for each of these constitutive models, including brittle fracture, have been derived.

While for linearized PD models and PD correspondence models of the form shown in [32], convolutional structures are easy to obtain (see [42–44]), a case-by-case investigation is needed for general nonlinear models to find a convolutional form to which FCBM can be applied. One example for a nonlinear bond-based model is provided in [23], while references [43, 44] show the procedure for obtaining the convolution form in the case of elastoplasticity and ductile failure.

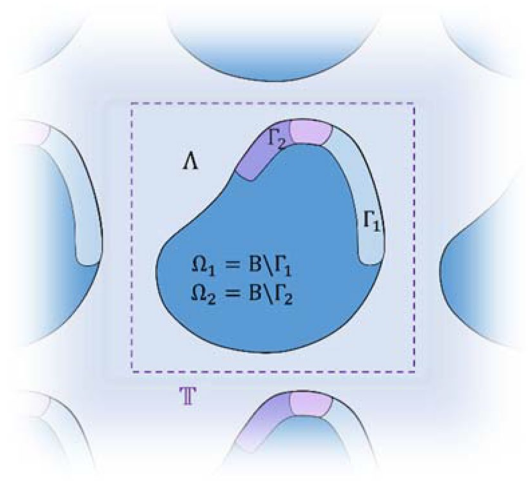
3 Review of the Fast Convolution-Based Discretization Method (FCBM)

PeriFast/Dynamics uses the fast convolution-based method (FCBM) to solve the PD-IVVC problem in Eq. (1). In FCBM, the convolution theorem and efficient FFT algorithms are employed to evaluate the mid-point quadrature at significantly lower costs compared to the direct summation that is traditionally used. Details of the method are given in [23] and briefly summarized below. Identification and looping over neighbors of a given node are no longer needed in FCBM, making the method independent of the neighbor numbers. The initial family search is eliminated, and memory allocation is significantly reduced, since neighbor information does not need to be stored. We aim to approximate the integral over the horizon region in Eq. (1) using mid-point integration (one-point Gaussian quadrature) but evaluated using the Fast Fourier Transform (FFT) and its inverse, instead of the regular direct summation through a nested loop over the horizon region. For FFT to be applicable in computing the convolution sums, the problem needs to be extended by periodicity to the entire space. This is done by first embedding the PD domain in a rectangular box (with a buffer of at least δ between the surface of the domain and the edge of the box), which is then extended by periodicity to the entire space.

Figure 2 shows the box (delineated by the dashed line) with the actual domain contained in it, extended by periodicity as depicted in Fig. 1. Note that the box edges should be at least one horizon (δ) away from the boundary of the body B. This will ensure that there will be no wrap-around effect in the circular convolution discussed below.

After extension of the body to $\mathbb{T}$, the following *characteristic functions*, are defined for distinguishing various subdomains (partitioning $\mathbb{T}$ in a way):

$$\chi_{\mathrm{B}}(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x} \in \mathrm{B} \\ 0 & \boldsymbol{x} \in \mathbb{T} \backslash \mathrm{B} = \Lambda \end{cases} \tag{7}$$



Γ_2
Λ
Γ_1
$\Omega_1 = \mathrm{B}\backslash\Gamma_1$
$\Omega_2 = \mathrm{B}\backslash\Gamma_2$
$\mathbb{T}$

Fig. 2 Extension of a generic peridynamic body to a periodic box in 2D. (Figure adopted from [23])

$$\chi_{\Omega_i}(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x} \in \Omega_i \\ 0 & \boldsymbol{x} \in \mathbb{T}\backslash\Omega_i = \Gamma_i \cup \Lambda \end{cases} ; \text{ and } \chi_{\Gamma_i}(\boldsymbol{x}) = \chi_{\mathrm{B}}(\boldsymbol{x}) - \chi_{\Omega_i}(\boldsymbol{x}) \tag{8}$$

χ_{B} is defined for eliminating any interaction between the PD body and the rest of the box and χ_{Ω_i} is for applying the BCs.

Using χ_{B} and χ_{Ω_i}, the PD IVVC problem in Eq. (1) is modified as follows:

$$\begin{cases} \rho\frac{\partial^2 u_i}{\partial t^2} = \chi_{\Omega_i}\left(\int_{\mathbb{T}}\chi_{\mathrm{B}}\chi'_{\mathrm{B}}\mu f_i \mathrm{d}V_{x'} + b_i\right) + \left(1-\chi_{\Omega_i}\right)\rho\frac{\partial^2 w_i}{\partial t^2} & \boldsymbol{x}\in\mathbb{T},\ t>0 \\ u_i(\boldsymbol{x},0) = \chi_{\Omega_i}u_i^0 + \left(1-\chi_{\Omega_i}\right)w_i(\boldsymbol{x},0) & \boldsymbol{x}\in\mathbb{T} \\ v_i(\boldsymbol{x},0) = \chi_{\Omega_i}v_i^0 + \left(1-\chi_{\Omega_i}\right)\frac{\partial w_i(\boldsymbol{x},0)}{\partial t} & \boldsymbol{x}\in\mathbb{T} \end{cases} \tag{9}$$

where $w_i(\boldsymbol{x},t)$ is known from the given data:

$$w_i(\boldsymbol{x},t) = \begin{cases} g_i(\boldsymbol{x},t) & \boldsymbol{x}\in\Gamma_i \\ 0 & \boldsymbol{x}\in\mathbb{T}\backslash\Gamma_i \end{cases} \tag{10}$$

Changing the domain of integration from $\mathcal{H}_x$ in Eq. (2) to $\mathbb{T}$ in Eq. (10) does not alter the integral because f_i is zero outside of the horizon region.

The solution to Eq. (10) on Ω_i is the same as the solution to Eq. (1). Equation (10), however, is defined over a periodic domain ($\mathbb{T}$) which allows for utilizing FFT for fast evaluation of the circular convolutions arising from discretization of PD integrals.

PeriFast/Dynamics uses uniform grid spacing for spatial discretization at this stage. The discrete coordinates are defined:

$$\begin{aligned} &\boldsymbol{x}_{nmp} = \left\{(n-1)\Delta x_1, (m-1)\Delta x_2, (p-1)\Delta x_3\right\}, \text{ where } \Delta x_1 = \tfrac{L_1}{N_1};\ \Delta x_2 = \tfrac{L_2}{N_2};\Delta x_3 = \tfrac{L_3}{N_3} \\ &\text{and } n = \left\{1,2,\ldots,N_1\right\};\ m = \left\{1,2,\ldots,N_2\right\}; p = \left\{1,2,\ldots,N_3\right\} \end{aligned} \tag{11}$$

L_1, L_2 and L_3 are the dimensions of the box $\mathbb{T}$ in 3D, and N_1, N_2, and N_3 are number of nodes in each coordinate direction. Note that FCBM might be compatible with nonuniform discretizations if the nonuniform FFT is employed, but this is an area for future research.

Using mid-point quadrature for the integral in Eq. (9). One gets

$$L_i^{nmp,t} = \sum_{q,s,r=1}^{N_3,N_2,N_1} \chi_{\mathrm{B}}^{nmp} \chi_{\mathrm{B}}^{rsq} \mu\left(\boldsymbol{x}_{rsq}, \boldsymbol{x}_{nmp}, t\right) f_i\left(\boldsymbol{x}_{rsq}, \boldsymbol{x}_{nmp}, t\right) V^{rsq} \tag{12}$$

where $\sum_{q,s,r=1}^{N_3,N_2,N_1} = \sum_{q=1}^{N_3} \sum_{s=1}^{N_2} \sum_{r=1}^{N_1}$. Note that to compute PD integrals more accurately, one can use the partial-volume correction algorithms [45, 46]. These algorithms can be easily incorporated to the FCBM framework by introducing a volume correction function to Eq. (12). The correction functions can be defined similar to the one defined in [45]. This is not done here because we will tend to use relatively large m-values (m is the ratio of horizon size to grid spacing), reducing the error in that way. An analysis of the influence of partial-volume algorithms on FBCM results is planned in the future.

The key step in FCBM is to express the summation in the equation above in terms of linear combinations of convolutions, in the following general form:

$$\begin{aligned} L_i^{nmp,t} &= \sum_{l=1}^{N_c} \left[a_l\left(\boldsymbol{x}_{nmp}, t\right) \sum_{q,s,r=1}^{N_3,N_2,N_1} b_l\left(\boldsymbol{x}_{rsq}, t\right) c_l\left(\boldsymbol{x}_{nmp} - \boldsymbol{x}_{rsq}, t\right) \right] \\ &= \sum_{l=1}^{N_c} a_l^{nmp,t} \left[b_l * c_l \right]^{nmp,t} \end{aligned} \tag{13}$$

where N_c is a positive integer that denotes the number of convolutions, and for each $l = 1, \ldots, N_c$: a_l is a function of point $\boldsymbol{x}$, b_l is function of $\boldsymbol{x}'$, and c_l is a function of $(\boldsymbol{x} - \boldsymbol{x}')$. Here c_l functions are referred to as the *kernel functions*. Note that different constitutive models lead to different a_l, b_l, and c_l functions that need to be defined in the code. Convolutional forms for the linearized bond-based, linearized native state-based, and PD correspondence models used in this work are provided in Appendix. Generally, a convolutional structure is natural for the integral operator in linear PD formulations [23, 47]. For nonlinear PD models, one needs to either linearize them or investigate on a case-by-case basis to see if such a structure can be found. In our previous publication [23] (see also Eq. (13) in the present manuscript) we showed how to obtain the convolutional structure for a large class of nonlinear PD problems. For problems that do not fall directly into this general setting, like the PD model with critical bond-strain damage criterion, we had to introduce a modified damage criterion (based on critical nodal strain energy density, instead of critical bond-strain) which allowed us to recast the formulation into that general setting and easily obtain the convolutional structure needed. Several examples for constructing convolution-based discretizations for nonlinear PD problems have been shown for nonlinear diffusion [22], and nonlinear elasticity (bond-based) with brittle fracture [23]. Notably, PD correspondence models of the form presented in [38] also fall into the general setting mentioned above and, therefore, it is easy to derive their convolutional structure (see [42–44]).

Using the discrete convolution theorem, Eq. (13) can be computed as

$$L_i^{nmp,t} = \sum_{l=1}^{N_c} a_l^{nmp,t} \left\{ \mathbf{F}^{-1} \left[\mathbf{F}\left(b_l\right) \mathbf{F}\left(c_l^s\right) \right] \right\}^{nmp,t} \tag{14}$$

where $\mathbf{F}$ and $\mathbf{F}^{-1}$ denote the FFT and inverse FFT operations, and c_l^s is the *shifted kernel* with respect to the box coordinates. c_l^s is the periodic version of c_l function over $\mathbb{T}$, where the origin of c_l is shifted to coincide with the corners of $\mathbb{T}$. This is necessary for the circular convolution operation to represent the PD convolution integrals. Figure 3 shows the original and the shifted version of a generic 2D radial kernel.

In Section 4.6 the operation of generating the shifted kernel from the given kernel function in PeriFast is described.

By comparing Eqs. (13) and (15), we can see that the summation over the neighbors of $\boldsymbol{x}_{nmp}$ no longer appears in the fast convolution computation, and therefore FCBM is independent of the number of neighbors of a given node. As a consequence, there is no need to search, identify, and store neighbor information, leading to important CPU and storage savings.

The displacement and velocity fields at each time step Δt are updated explicitly via the velocity-Verlet algorithm (see [4] for details):

$$u_i^{nmp,t+\Delta t} = \chi_{\Omega_i}^{nmp}\left\{u_i^{nmp,t} + \Delta t\left[v_i^{nmp,t} + \frac{\Delta t}{2\rho}\left(L_i^{nmp,t} + b_i^{nmp,t}\right)\right]\right\} + \left(1 - \chi_{\Omega_i}^{nmp}\right)w_i^{nmp,t+\Delta t} \tag{15}$$

$$v_i^{nmp,t+\Delta t} = \chi_{\Omega_i}^{nmp}\left\{v_i^{nmp,t} + \frac{\Delta t}{2\rho}\left[\left(L_i^{nmp,t} + b_i^{nmp,t}\right) + \left(L_i^{nmp,t+\Delta t} + b_i^{nmp,t+\Delta t}\right)\right]\right\} \tag{16}$$

Remark: in addition to the internal force density, all other PD integrals, if used (e.g., PD strain energy density), need to be expressed in the form of Eq. (13), in order to be computed using the FCBM.

Remark: in order to impose periodic BC in FCBM, one takes $\chi_{\mathrm{B}}(\boldsymbol{x}) = \chi_{\Omega_i}(\boldsymbol{x}) = 1$ for all $\boldsymbol{x}$. This implies that the body becomes a torus/periodic box: $\chi_{\mathrm{B}} \equiv \chi_{\Omega_i} \equiv \mathbb{T}$. In this case, the Fourier basis functions employed in the FFT operations naturally capture the "wrap-around" effect expected in a periodic setting. This is in contrast with other discretization methods such as FEM or other meshfree methods where the periodic/wrap-around condition needs to be explicitly enforced on the boundary nodes. In the case of Fourier-based discretizations, like Fourier spectral methods and FCBM, periodic BCs are naturally captured and there is no need to explicitly enforce any type of conditions or constrains. The characteristic function

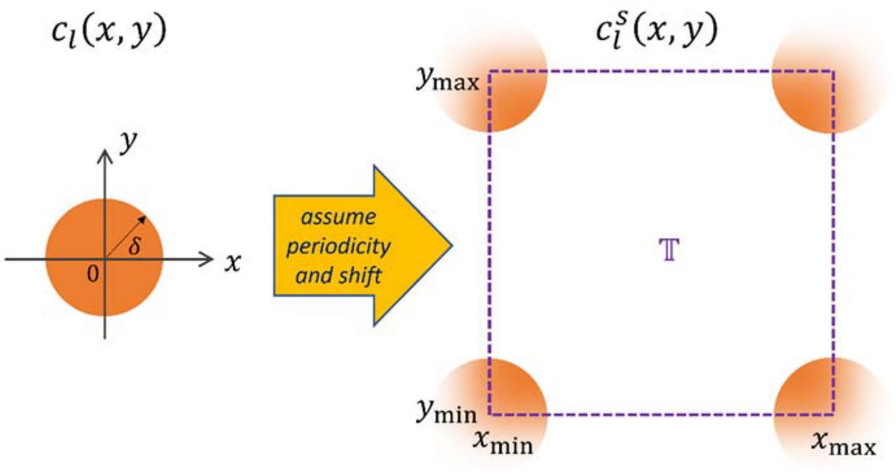


Fig. 3 On the left: a 2D generic kernel function in its original form centered at zero (c_l). On the right: the periodic shifted version (c_l^s) used in the fast convolution on $\mathbb{T} = [x_{\min}x_{\max}] \times [y_{\min}y_{\max}]$. The colored disk denotes the non-zero part of the kernel function

introduced in our FCBM approach allows the extension of this Fourier-based method to bounded domains with non-periodic BCs.

Remark: convergence studies for FCBM have shown a quadratic rate for diffusion problems in 1D and 2D [21, 22] and a super linear rate in a 3D elasticity example [23]. For fracture problems, an m-convergence study in 2D was reported in [23], and a δ-convergence study can be found in [43, 44]. Convergence studies for certain variations of FCBM can also be found in [24–26].

4 PeriFast/Dynamics Code Description

In this section, we describe the data structures used in the discretization in PeriFast/Dynamics, discuss the overall structure of the code and provide details of each of its modules (m-files).

4.1 Data Structure for PD Nodes

PeriFast/Dynamics stores the PD nodal positions and nodal values for different quantities in a consistent way with MATLAB's multi-dimensional FFT operations.

Let $\mathbb{T} = [x_{\min} x_{\max}] \times [y_{\min} y_{\max}] \times [z_{\min} z_{\max}]$ denote a periodic box in 3D, with the uniform discretization given below:

$$\begin{aligned}&\boldsymbol{x}_{ijk} = \begin{Bmatrix} x_i \\ y_j \\ z_k \end{Bmatrix} = \begin{Bmatrix} x_{\min} + (i-1)\Delta x \\ y_{\min} + (j-1)\Delta y \\ z_{\min} + (k-1)\Delta z \end{Bmatrix}; \Delta x = \tfrac{L_1}{N_1}, \Delta y = \tfrac{L_2}{N_2}; \Delta z = \tfrac{L_3}{N_3}; \\ &\text{and } i = 1, 2, \ldots, N_1 j = 1, 2, \ldots, N_2 k = 1, 2, \ldots, N_3, \\ &L_1 = x_{max} - x_{min}, L_2 = y_{max} - y_{min}, L_3 = z_{max} - z_{min}\end{aligned} \tag{17}$$

In PeriFast/Dynamics the x, y, and z-coordinates of all nodes are stored in three distinct 3D arrays of size $N_2 \times N_1 \times N_3$:

$$\begin{aligned}&\mathbf{X} = \left[X_{jik}\right]_{N_2 \times N_1 \times N_3}, \mathbf{Y} = \left[Y_{jik}\right]_{N_2 \times N_1 \times N_3}, \mathbf{Z} = \left[Z_{jik}\right]_{N_2 \times N_1 \times N_3} \\ &i = 1, 2, \ldots, N_1 j = 1, 2, \ldots, N_2 k = 1, 2, \ldots, N_3\end{aligned} \tag{18}$$

where X_{jik}, Y_{jik}, and Z_{jik} respectively denote the x, y, and z-coordinates of node $\boldsymbol{x}_{ijk}$. Note that the index in y-direction precedes the x-direction index, due to the way MATLAB's *meshgrid* function generates 3D arrays. While in traditional solvers all nodal data are usually vectorized regardless of the spatial dimension, in FCBM it is necessary to work with multi-dimensional arrays, because of using multi-dimensional FFT operations.

In PeriFast/Dynamics, functions of space and time are defined as functions of **X**,**Y**, **Z**, and t, and return outputs in the form of 3D $N_2 \times N_1 \times N_3$ arrays containing their nodal values.

For example, let $\boldsymbol{C}(\boldsymbol{x})$ be a 3×3 tensor-valued function defined in 3D. The discrete version of this function in PeriFast/Dynamics is

$$\boldsymbol{C}_{pq}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \left[\left(C_{pq}\right)_{jik}\right]_{N_2 \times N_1 \times N_3}; \text{for each } p, q = 1, 2, 3 \tag{19}$$

For each p and q (each component of the tensor $\boldsymbol{C}$), $\mathbf{C}_{pq}$ is a 3D $N_2 \times N_1 \times N_3$ array returned by a function of **X**,**Y**, and **Z**. See PeriFast's *nodes_and_sets.m* module for examples of such definitions.

4.2 The Overall Code Structure

The current version of PeriFast/Dynamics consists of 14 MATLAB m-files: *main.m, inputs.m, nodes_and_sets.m, pre_constitutive.m, constitutive.m, update_tractions.m, update_VC.m, initial_gpu_arrays.m, dump_output.m, visualization.m, open_Matlab_video.m, create_matlab_video.m close_Matlab_video.m and postprocess.m.main.m* is the script that executes the program. *inputs.m* contains certain input data including material properties, simulation time, time steps, initial and boundary conditions, visualization parameters. *nodes_and_sets.m* contains the PD horizon and discrete geometrical data including nodal coordinates and discrete characteristic functions that define various subdomains: the original body, constrained volumes, pre-damaged regions, and subregions where tractions are applied as a body force. *Pre_constitutive.m*, and *constitutive.m* contain the material model information (available in the form of Eq. (13)). Functions that are independent of the field variables and time, i.e., are not changing during the simulations, are defined in *pre_constitutive.m*. The kernel functions are usually of this type and are defined in this module. The precomputed functions in *pre_constitutive.m* as well as the displacement field and other inputs are passed onto the module *constitutive.m*, where the internal force density, strain energy density, and damage are computed. *constitutive.m* is the module that is called in each time step to update the material response. Files *update_tractions.m*, and *update_VC.m* are modules called when traction and displacement boundary conditions need to be updated, respectively. *initial_gpu_arrays.m,* converts variables involved in the convolution operations to MATLAB's "gpuarray" type to use GPU-based computations. *Dump_output.m* script is called every several time steps (frequency defined by the user in *inputs.m*) to record output data into a Matlab variable and into a Tecplot 360 [48] file (user can determine in *inputs.m* if Tecplot file is desired). If visualization is requested by the user (in *inputs.m*), *visualization.m* is called as well to plot results in Matlab at every snapshot during the analysis (number of data dump and visualization frames can be set by the user in *inputs.m).*

Naturally, performing the visualization during the analysis slows down the run time. For speed tests, or if solving a larger problem, it is recommended to turn off visualization_during_analysis (in *inputs.m*). One can postprocess the recorded output data once the simulation is completed. The option to generate a Tecplot output (tecplot_output in *inputs.m*) file may also affect the speed of the analysis.

For a given problem, the user needs to specify input data in *inputs.m* and geometrical data in *nodes_and_sets.m*. Currently, the geometry data (the characteristic function, the boundary regions, the box domain coordinates) is setup manually, on a case by case basis. The users are invited to contribute functions to the code that would automate this step, for example, to directly import various CAD systems representations of the geometry data.

In this version, three material models have been implemented in PeriFast/Dynamics: (1) linearized bond-based isotropic elastic; (2) linearized state-based isotropic elastic; and (3) PD correspondence model for a hyperelastic material. We model brittle damage in all of the three cases (see Section 2 and Appendix for the damage models). New material models can be added to PeriFast/Dynamics by defining additional material types in *pre_constitutive.m* and *constitutive.m*. The user can also easily specify additional variables to output in *dump_output.m* (e.g., internal variables in history dependent material models) and customize *visualization.m, open_Matlab_video.m, create_matlab_video.m* and *close_Matlab_video.m* as desired.

In the following, we take a closer look at each m-file.

4.3 Description of main.m

Box 1 shows the structure of *main.m* as the executable main file of the program. The file consists of three stages: reading input information, initialization, and the time computation loop. After the time loop a outputs are saved in a file named: results.mat.

4.4 Description of inputs.m

In the *inputs.m* file, user-prescribed data are assigned to variables and passed onto the main program. The user needs to directly insert the input data in this file.

The terms in the parentheses denote the MATLAB variable names used in the code. *Props* is a 1D array, while *Fb*, *IC_u* and *IC_v* are structure arrays-type variables, each containing three functions corresponding to each vector component. The function of the body force's x-component for example is *Fb(1).func*. The variables used for traction and displacement boundary conditions are also of struct type. For BCs, however, each coordinate direction has a distinct variable associated with it, containing the number of the prescribed BCs in that direction and the corresponding functions. For example, if one needs to enforce two traction BCs in y direction, one sets *trac_y.No* = *2*, and then define the two functions *trac_y(1).func* and *trac_y(2).func*.

The desired number of data dumps and frames for visualization is selected by the user through variables *number_of_data_dump* and *number_of_visualization_frames.*

Variable *tecplot_output* can be either 1 or 0. Choosing value 1 leads to the selected outputs being saved as a Tecplot file during the analysis (in the current version of the code we choose damage index as an output to be saved in a Tecplot file as an example; users can select any desired outputs). Using value 0 cancels the Tecplot output.

```
Call inputs.m function to generate the input data
Call nodes_and_sets.m function to generate geometrical data
Compute and store invariant functions of the material model (call pre_constitutive.m, see Section 4.6)
    -   Initialize the time loop:
    -   Assign initial conditions (using output data from inputs.m)
    -   Assign pre-damage (if any) (using data generated by nodes_and_sets.m)
    -   Compute initial body force (using output of inputs.m)
    -   Compute initial traction forces using update_tractions.m (see Section 4.9)
    -   Compute initial internal force density using constitutive.m (see Section 4.7)
    -   If GPU option is selected in inputs.m, setup GPU arrays using initial_gpu_arryas.m (see section
        4.10)
 For each time step:
    -   Update volume constraints using update_VC.m (see Section 4.8)
    -   Update displacement field (via velocity-Verlet method with embedded constraint in Eq. (15))
    -   Update internal force using constitutive.m (see Section 4.7)
    -   Update tractions using update_tractions.m (see Section 4.9)
    -   Update body force
    -   Update velocity field using Eq. (16)
    -   Dump output data and visualize the results using dump_output.m and visualization.m (see
        Section  4.11 and 4.12) if visualization option is on in inputs.m.
End time loop
    -   Save outputs to a .mat file.
```

Box 1 Structure of the main program file: *main.m*

Define the following variables and return as the function output:
material properties (*props*)
simulation time (*t_max*)
time step (*dt*)
GPU run switch (run_in_gpu)
number of dump data (*number_of_data_dump*)
visulalization frames frequency (number_of_visualization_frames)
Tecplot switch (*tecplot_output*)
visualization during analysis switch (*visualization_during_analysis*) select desired outputs for being plotted/animated (*outputs_var_for_visulazation*)
body force density (*Fb*)
initial conditions:
- displacements as functions of space (*IC_u*)
- velocities as functions of space (*IC_v*)

traction boundary conditions (*trac_x, trac_y, trac_z*):
- number of tractions
- each traction as a function of space and time

displacement boundary conditions (*dispBC_x, dispBC_y, dispBC_z*):
- number of displacement BCs
- each displacement BC as a function of space and time

Box 2 Structure of *inputs.m*

Variable *visualization_during_analysis*is either 0 or 1, with 1 requesting Matlab visualization during the analysis phase, and 0 leaving out run-time visualization. Note that plots/animations can be obtained by postprocessing the output saved in R*esults. mat* file. The variable *visualization_during_analysis* can be set to 0 so that the results can be plotted/animated by running *postprocess.m* after saving *Results.mat*. This is, in fact, the recommended option when solving larger problems since plotting during analysis slows down the solver. In this version of the code the default option sets the *visualization_during_analysis*is variable to 0.

The user can choose the desired outputs among dumped outputs (*dump_output.m*) to be plotted/animated by defining a vector of integers, *outputs_var_for_visualization* in *inputs.m*. A number is assigned for each output and used for defining *outputs_var_for_visualization* in *inputs.m*. In the current version of PeriFast/Dynamic code, u_1,u_2,u_3,*u_mag*,v_1,v_2,v_3, *v_mag, W, d* (i.e., displacement vector components, displacement magnitude, velocity vector components, velocity magnitude, strain energy density, damage index) and *lambda* are dumped as outputs and the assigned number for them is 1 through 11, respectively. For example, if users want to visualize u_1, u_2, and d among these dumped outputs, they need to set *outputs_var_for_visualization* = [1, 2, 10]. Note that in order to add any other outputs for visualization which is not defined in the current version of *dump. output.m,* users first need to add it to *dump.output.m* and then modify *visualization.m, open_Matlab_video.m, create_Matlab_video.m and close_Matlab_video.m* to visualize that as well.

4.5 Description of nodes_and_sets.m

nodes_and_sets.m contains nodal coordinates and the geometrical information of the problem. Before describing its details, we first point out how the domain extension required by

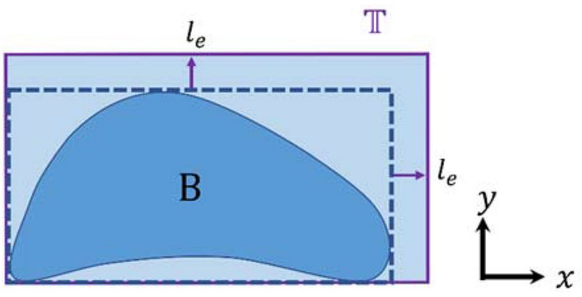


Fig. 4 A 2D generic PD body (B), the enclosing box (shown by the dash-line), and one possible extension to the periodic box $\mathbb{T}$

FCBM (see Fig. 2) is implemented in PeriFast/Dynamics. Given a PD body (B) defined by the original PD-IVVC problem, it is first assumed that the body is enclosed in a rectangular box, as tight as possible to the body. This enclosing box is shown in Fig. 4 for the 2D case with dash-line. Note that this is different from the box that is repeated by periodicity. Assuming a coordinate origin, we define the coordinates of the enclosing box vertices. To construct the periodic box $\mathbb{T}$, the enclosing box is extended along each direction/axis with an extension *at least as large as the horizon size* to avoid the "wrap-around" effect in the circular convolution. Figure 4 shows a PD body, the enclosing box (the dash-line), and the extended periodic box $\mathbb{T}$ in 2D. l_e denotes the extension length (which should be selected larger than δ).

Note that the best choice of the enclosing box (and the coordinate system directions) is the one that leads to the least extra space between the body and the box. Considering a fixed nodal spacing, less gap results in less excess degrees of freedom in FCBM. If the body itself is a rectangular box, then the enclosing box would be the body itself.

Box 3 shows the structure of *nodes_and_sets.m*.

In *nodes_and_sets.m*, the user first defines the horizon size (δ), the enclosing box dimensions, the extension length (l_e in Fig. 4), and the number of nodes in each direction. The program then extends the enclosed box to find $\mathbb{T}$, and then create nodes according to Eqs. (14) and (15). Next, the various characteristic functions/node sets are defined by the user to describe different subdomains corresponding to the original body, traction forces, volume constraints, and pre-damage. At the end, node sets representing displacement BCs in the same directions are merged to form three distinct node sets $\chi_{\Gamma_1}, \chi_{\Gamma_2}, \chi_{\Gamma_3}$. Then $\chi_{\Omega_1}, \chi_{\Omega_2}, \chi_{\Omega_3}$ are obtained by Eq. (5). The horizon, box $\mathbb{T}$ info, nodal coordinates, and

```
Define PD horizon size (delta)
Define the enclosing box (x_min, x_max, y_min, y_max, z_min, z_max)
Define the extension length (extension)
Define the number of nodes in each direction (Nx, Ny, Nz)
Extend the enclosing box to the periodic box
Create nodes and nodal volumes
Construct the characteristic functions (i.e., node sets) for:
    -   original body (chiB)
    -   traction BC (chit_x, chit_y, chit_z)
    -   displacement BC (chiG_x, chiG_y, chiG_z)
    -   pre-damage (chi_predam)
Assemble $\chi_{\Gamma_i}$ (chiGx, chiGy, chiGz)
Compute and return $\chi_{\Omega_i}$ (chiOx, chiOy, chiOz)
```

Box 3 Structure of *nodes_and_sets.m*

the characteristic functions are passed onto *main.m* to be used in the analysis.*chit_x, chit_y, chit_z*, and *chiG_x, chiG_y, chiG_z* are all struct type variables and include the number of BCs in their specific direction, as well as the node sets for each of those. For example, if there are two traction BCs given in the y direction, one needs to set *chit_y.No* = *2*, and define *chit_y(1).set* and *chit_y(2).set*, where each of these sets are 3D $N_2 \times N_1 \times N_3$ arrays with value 1 for nodes in the node set and zero elsewhere.

Remark: the number of node sets in *chit_x, chit_y, chit_z*, and *chiG_x, chiG_y, chiG_z*, should be consistent with number of tractions and displacement BCs given by *trac_x, trac_y, trac_z*, and *dispBC_x, dispBC_y, dispBC_z*, in *inputs.m* respectively.

4.6 Description of pre_constitutive.m

This m-file contains the time-invariant functions needed for evaluation of the PD constitutive terms such as the internal force, strain energy, etc., available in the form of form of Eqs. (10) and (11). For most well-known material models, kernel functions (c_l is Eq. (10)) are invariant in time and should be defined in this module. Note that this module returns the FFT of the kernel functions in their shifted forms (c_l^s) described in previous section (see Fig. 3).

Box 4 gives the structure of *pre_constitutive.m*.

Here is how the “shift operation” shown in Fig. 4 is carried out in PeriFast/Dynamics: for obtaining c_l^s, first, c_l is translated such that its origin coincides with the center of the box: $c_l\left(\mathbf{X}-x_c, \mathbf{Y}-y_c, \mathbf{Z}-z_c\right)$. Then the *fftshift* MATLAB function is used on the translated c_l. The *fftshift* command breaks down the array from mid-planes of the box and swap the partitions, resulting in the desired shifted form: c_l^s. More information on *fftshift* is provided in the MATLAB documentation.

The coded PD correspondence model for the hyperelastic material (material ID = 2) uses St. Venant–Kirchhoff classical model for finite deformation elasticity. The implemented correspondence model includes the stability term introduced in [33] to suppresses zero energy modes and stabilizing the PD correspondence solutions.

4.7 Description of constitutive.m

This module takes the displacement field, history-dependent variables such as the old damage parameter, material properties (defined in *inputs.m*), discretization info (defined in

```
Module inputs: material properties, horizon, nodal coordinates, periodic box dimensions
Read material ID from the properties
Calculate the coordinates of the center of the box (x_c, y_c, z_c)
If material ID = 0 (Linearized bond-based elastic material)
    -   Compute the PD elasticity constants from material properties
    -   Define kernel functions $c_l(\mathbf{X},\mathbf{Y},\mathbf{Z})$ (see Eq. (13))
    -   Perform shift operation on $c_l$ to obtain $c_l^s$
    -   Compute FFT of $c_l^s$ functions: $\widehat{c_l^s}$
    -   Return $\widehat{c_l^s}$ as the module output
If material ID = 1 (Linearized state-based elastic material)
    -   [same procedure as BB, but with different functions]
If material ID = 2 (PD correspondence model for hyperelastic material)
    -   [same procedure as BB, but with different functions]
```

Box 4 Structure of *pre_constitutive.m*

nodes_and_sets.m), and the invariant parameters in the constitutive response (from *pre_constitutive.m*) as inputs, and returns the internal force density, strain energy density, and updated history-dependent variables (e.g. damage) as outputs. Box 5 presents the structure of this module.

Note that user-defined material models are allowed in PeriFast/Dynamics and can be introduced by defining appropriate functions in *pre_constitutive.m* and *constitutive.m*, with additional material IDs, in the If-statements.

While PeriFast/Dynamics can adopt different user-defined damage models along with the user-defined constitutive laws, in the current version, for the three included constitutive models, we used the same energy-based pointwise damage model introduced in [23]. In this damage model, the parameter that store damage information is a binary variable denoted by *lambda* which is 0 for a damaged node and 1 otherwise. A damaged node is a node for which its strain energy density exceeds a threshold calibrated to the critical fracture energy of the material. The damage index (here tracked by the variable named *damage*) varies between 0 and 1 and it is computed from *lambda* using the following relation [23]:

$$d(\boldsymbol{x},t) = 1 - \frac{\int_{\mathcal{H}_x} \chi_{\mathrm{B}} \chi'_B \mu \mathrm{d}V_{x'}}{\int_{\mathcal{H}_x} \chi_{\mathrm{B}} \chi'_B \mathrm{d}V_{x'}} = 1 - \frac{\int_{\mathcal{H}_x} \chi_{\mathrm{B}} \chi'_B \lambda\lambda' \omega(|\boldsymbol{\xi}|) \mathrm{d}V_{x'}}{\int_{\mathcal{H}_x} \chi_{\mathrm{B}} \chi'_B \omega(|\boldsymbol{\xi}|) \mathrm{d}V_{x'}} \tag{20}$$

In Eq. (20), ω is the influence function and $\boldsymbol{\xi} = \boldsymbol{x'} - \boldsymbol{x}$ denotes the bond vector. The influence function $\omega(|\boldsymbol{\xi}|) = \frac{1}{|\xi|}$ is used in this work.

Lambda, *damage,* and any other history-dependent quantities, are defined in a structure-type variable named *history_var*.

Remark: if one intends to study stress waves only (i.e., deformation without damage/fracture), one can either comment out the commands corresponding to updating damage, or just prescribe a very large fracture energy value in *inputs.m*.

4.8 Description of update_VC.m

This module takes the displacement BCs as functions of space and time (from *inputs.m*), and also their corresponding node sets (from *nodes_and_sets.m*), and returns the nodal values for functions w_i ($i = 1, 2, 3$) in Eq. (15) as outputs. Box 6 shows the structure of *update_VC.m*.

```
Module inputs: material properties, displacements, history-dependent variables, horizon, body node
set, nodal coordinates, nodal volume
Read material ID from the properties
If material ID = 0 (Linearized bond-based elastic material)
    -   Compute the frequently repeated terms, and store for the following computations
    -   Compute the internal force density: (L1, L2, L3) (from Eq. (14))
    -   Compute strain energy density (W)
    -   Update the damage parameter (lambda) given the old lambda and W
    -   Compute the damage index (damage)
    -   Return L1, L2, L3, W, damage, lambda
If material ID = 1 (Linearized state-based elastic material)
    -   [same procedure as BB, but with different functions]
If material ID = 2 (PD correspondence model for hyperelastic material)
    -   [same procedure as BB, but with different functions]
```

Box 5 Structure of *constitutive.m*

```
Module inputs: functions of displacement BCs, their node sets, nodal coordinates, time
Assemble $w_1$ (wx) using displacement functions in x (dispBC_x) and their node sets (chiG_x)
Assemble $w_2$ (wy) using displacement functions in y (dispBC_y) and their node sets (chiG_y)
Assemble $w_3$ (wz) using displacement functions in z (dispBC_z) and their node sets (chiG_z)
Return assembled volume constraints (wx, wy, wz)
```

Box 6 Structure of *update_VC.m*

4.9 Description of update_tractions.m

In PeriFast/Dynamics, traction BCs are enforced as body forces applied uniformly on a δ -thick layer of the body at the boundary (distributed uniformly through the thickness of the layer). The body force nodal value is obtained by dividing the traction force at a point by δ. The structure of *update_tractions.m* is very similar to *update_VC.m.*

4.10 Description of initial_gpu_array.m

To accelerate computations using GPUs, one needs to convert variables involved in the convolution operations to MATLAB's "gpuarray" type using the file *initial_gpu_array.m*. Then, calls to MATLAB's FFT and inverse FFT functions will automatically use the GPU for these operations. Note that the Parallel Computing Toolbox needs to be installed to enable GPU computing in MATLAB.

4.11 Description of dump_output.m

This module gets the snapshot number (*ks*), displacements and velocities in x, y, and z directions, strain energy density, damage index, and *lambda* as inputs. These variables along with other post-processed quantities such as displacement magnitude, are stored in a single structure-type MATLAB variable named *Output*. If the visualization switch is on (if *visualization_during_analysis* $==1$) this variable is passed onto the visualization module for creating MATLAB plots during the analysis. The frequency of visualization of outputs is dependent on the *number _of _visualization_frames* defined in *inputs.m.* Also, if the Tecplot switch is ON in *inputs.m* file, the desired output is saved as a Tecplot file (.plt). Results stored in *Output* can be used for any desired post-processing operation. *dump_output.m* module can be easily modified by the user to store other user-defined outputs.

4.12 Description of visualization.m and postprocess.m

This module takes the outputs from *dump_output.m*, the snapshot number, nodal coordinates, and the body node set, and uses them to visualize the results. This module, too, can be easily

```
Module inputs: functions of traction BCs, their node sets, horizon, nodal coordinates, time
Assemble $b_1$ (btx) using traction functions in x (trac_x), their node sets (chit_x), and horizon (delta)
Assemble $b_2$ (bty) using traction functions in y (trac_y), their node sets (chit_y), and horizon (delta)
Assemble $b_3$ (btz) using traction functions in z (trac_z), their node sets (chit_z), and horizon (delta)
Return assembled traction body force (btx, bty, btz)
```

Box 7 Structure of *update_tractions.m*

modified by the user to plot the desired figures and/or record animations (user can select the desired output for visualization in *inputs.m*), and to export files in user-defined formats for further processing in external software. In order to record Matlab videos from the snapshots, *create_Matlab_video.m* is used. There is an option in *input.m* to select whether the user desires to visualize the results during the analysis or after. The default is to perform the visualization after the analysis by running *postprocess.m* and using the data saved in the Results.mat file.

4.13 Description of open_Matlab_video.m, create_Matlab_video.m and close_Matlab_video.m

These modules are used for creating Matlab videos from the outputs. For every desired output to be animated, first, a video file needs to be opened using *open_Matlab_video.m*. Next, by calling *create_Matlab_video.m*, the sequence of frames from the desired output is written to the video file. Finally, the video file needs to be closed by using *close_Matlab_video.m*. In the current version of PeriFast/Dynamics, a video file for damage evolution is created. Users can easily add any other desired output for creating a video by modifying *outputs_var_for_visualization* in *inputs.m*. For example, for the nodal velocity vector components and the strain energy density, one can define *outputs_var_for_visualization* = [5–7, 9], where v_1, v_2, v_3, and W are assigned the indices 5, 6, 7, and 9 in this version of code.

5 Example of Running PeriFast/Dynamics: 3D Dynamic Analysis of Brittle Fracture in a Glass Plate

In this section, we show how a particular problem on dynamic fracture in glass is setup and run with PeriFast/Dynamics. The physical problem is an example of dynamic brittle fracture in which crack branching takes place, when the applied loading is sufficiently high. For the crack to grow straight, one needs to lower the applied stress, see below. These type of problems, until the advent of PD, have been especially difficult to correctly simulate [12, 49].

5.1 Problem Setup

We consider a thin single-edge glass plate of size $0.1 \times 0.04 \times 0.002\ m^3$ with a pre-crack, subjected to sudden uniaxial tensile stress of $\sigma_0 = 4$ MPa on its top and bottom edges (see Fig. 5). These types of boundary conditions are not easily replicated in experiments, with crack surface ramped-up loadings being a more realizable scenario [50]. However, these

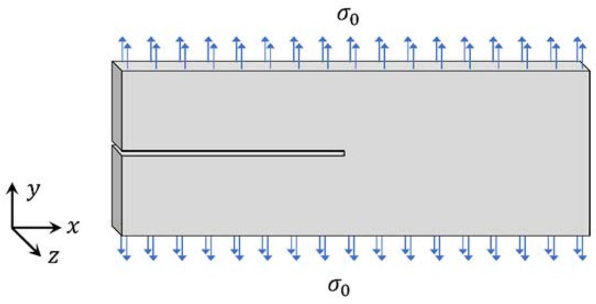


Fig. 5 Problem description for the 3D numerical example of dynamic brittle fracture. The thickness of the sample along the z-direction is exaggerated for visibility

boundary conditions are the most employed in numerical simulations of crack branching, and this is the reason for using them here. See [49] for comparison of different types of dynamic loading that induce crack branching in glass samples of this type.

The material properties are selected the same as in [49]: density $\rho = 2440\ kg.m^{-3}$, Young modulus$E = 72$ GPa, Poisson ratio $\nu = 0.25$, and fracture energy $G_0 = 3.8\ J.m^{-2}$. Since ν is restricted to be 0.25 for the bond-based model, the choice of 0.25 with the state-based models, allows for the comparison of the simulation results against each other. The horizon size $\delta = 1.02$e-3 m, and the grid spacing is chosen to be $\Delta x = \Delta y = \Delta z = 2$e-4 m. The addition of 0.02e-3 to 1e-3 for the horizon size is to avoid numerical sensitivity when the horizon size δ is an exact multiple of the grid spacing.

5.2 Defining the Code Input Data

The following input data is used to solve the problem define above with PeriFast/Dynamics:

5.2.1 In inputs.m

1. Material properties are entries in the *props* variable in the following order: material ID, ρ, G_0, E, ν. Set:
   props = [0; 2440; 3.8; 72e9; 0.24];
   0, 1, or 2 for the material ID determines if which material model is employed. If material ID = 0, then the bond-based model is used, and the value defined for ν is disregarded.
2. Define the simulation time, time step, run in gpu switch, snapshot frequency, visualization, and Tecplot output switches as
   t_max = 33e-6;
   dt = 5e-8;
   number_of_data_dump= 100;
   number_of_visualization_frames = 30;
   run_in_gpu = 0;
   tecplot_output= 0;
   visualization_during_analysis= 0;
   outputs_var_for_visualization = [10];
3. Set all the functions for the body force components and the initial displacements and velocities to zero (this is the default case).
4. Define the traction BCs by setting
   trac_y.No = 2;
   trac_y(1).func = @(x,y,z,t) 4e6;
   trac_y(2).func = @(x,y,z,t) -4e6;

5.2.2 In nodes_and_sets.m:

1. Define the PD horizon size: delta = 1.02e-3;
2. Define the enclosing box by providing the minimum and the maximum of the box dimension along each coordinate direction in
   x_min = 0; x_max = 0.1;
   y_min = 0; y_max = 0.04;
   z_min = 0; z_max = 0.002;
3. Define the extension length to form the periodic box: extension = 2e-3;

4. As noted earlier, this should be larger than the horizon size.
   Define the resolution in each direction
   Nx = 510;
   Ny = 210;
   Nz = 20;
   which results in over 2 million nodes. These values are calculated from the dimension of the extended box and the grid spacing of $\Delta x = \Delta y = \Delta z = 2.5e\text{-}4$.
5. Define χ_B (body node set) with value 1 for nodes inside the body and 0 otherwise. For our example, we set
   chiB = ones (Ny, Nx, Nz);
   chiB (Z < z_min | Z > z_max) = 0;
   chiB (Y < y_min | Y > y_max) = 0;
   chiB (X < x_min | X > x_max) = 0;
6. Define the node sets for the traction BCs
   chit_y.No = 2;
   chit_y(1).set = double(chiB == 1 & Y > y_max - delta + dy/2 & Y < y_max + dy/2);
   chit_y(2).set = double(chiB == 1 & Y > y_min - dy/2 & Y < y_min + delta - dy/2);
   The "double" command converts the logical arrays in the arguments to double arrays for computation.
7. Define the node set representing the pre-crack (the notch region)
   chi_predam = double(chiB == 1 & abs(Y - y_min - Ldy/2)<= (delta/2) & X < x_min + Ldx/2);

5.3 Selection of Outputs

Components of the displacement and the velocity vectors as well as their magnitude, strain energy density, damage index, and lambda are selected as the output variables by defining the following commands in *dump_output.m* file:

```
Output(ks).u1=u1;
Output(ks).u2=u2;
Output(ks).u3=u3;
Output(ks).u_mag=sqrt(u1.^2+u2.^2+u3.^2);
Output(ks).v1=v1;
Output(ks).v2=v2;
Output(ks).v3=v3;
Output(ks).v_mag=sqrt(v1.^2+v2.^2+v3.^2);
Output(ks).W=W;
Output(ks).d=damage;
Output(ks).lambda=lambda;
```

5.4 Execution of the Program

For faster computations, one can use the Matlab's capabilities for parallel, multi-threading, and GPU-based computations. In the current version of PeriFast/Dynamics, multithreading is immediately accessed by simply changing the maximum number of threads used in the run: LASTN = maxNumCompThreads (*p*), at the beginning of *main.m*, where *p* is the max number of threads desired. The default option is for serial computations, using $p=1$. GPU-based computation is explained in Section 4.2.

LASTN = maxNumCompThreads(1);

To execute the code, we run the *main.m* file.

5.5 Visualization of Results

Figure 6 shows the damage index 3D MATLAB profiles obtained by the bond-based and the state- models (native and correspondence). Evolution of velocity fields, as well as strain energy density and damage index during fracture are provided in Videos 1, 2, and 3 for these PD models, respectively.

5.6 Explanations of Differences Between Models

The results shown in Fig. 6a for the bond-based model are similar to those obtained with a 2D plane stress simulation in [49]. This is a good verification of the PeriFast/Dynamics' implementation. The slight differences between damage patterns (branching near the edge) from the three FCBM-based models stem from the small actual difference between the PD constitutive models.

Although the force density in the state-based and the bond-based models in Eqs. (A-1 and A-5) are different in general, for the linearized versions in Eqs. (A-1 and A-6, if the Poisson ratio is chosen as ¼ in the state-based model, the first term in Eq. (A-6) vanishes and the bond-based formula is recovered, for points in the bulk. These models, however, even for the one-quarter Poisson ratio value, are slightly different near surfaces. The root cause for this difference is in the different PD elastic micro-moduli computed in these two models. In the bond-based formulation (see [47]) the micro-modulus is computed based on a calibration for points in the bulk, and, assuming no surface correction is used for points near boundaries, has the value equal to $\frac{12E}{\pi\delta^4}$ in 3D. In the state-based formulation, the bond-level elasticity constant, $\frac{30\mu}{m}$, depends on the *weighted volume* at a node, denoted by m. The weighted volume in our model is obtained numerically by approximating the following integral over the horizon (see [47]):

$$m = \int_{\mathcal{H}_x} \omega(|\xi|)|\xi|^2 \mathrm{dV}_{x'} \tag{21}$$

We can easily show the equivalency of the elastic constants in the native state-based model ($\frac{30\mu}{m}$) to the bond based micromoduli at the continuum level for points in the bulk by computing m for nodes in the bulk (over a full spherical neighborhood) and using the following influence function $\omega(|\xi|) = \frac{1}{|\xi|}$(used in this work):

$$\frac{30\mu}{m} = \frac{30\mu}{\int_0^{\pi}\int_0^{2\pi}\int_0^{\delta} r^3 sin\phi \; dr \; d\phi d\theta} = \frac{30\mu}{\pi\delta^4} = \frac{12E}{\pi\delta^4} \tag{22}$$

The domain of integration in computing m, i.e., neighborhood $\mathcal{H}_x$, varies, however, for nodes near surfaces, including original domain boundaries and growing crack surfaces, compared to the nodes in the bulk, leading to automatically modified bond-level elastic properties near the surfaces for the native state-based models. In other words, for points near the boundary, the function m, according to Eq. (21) in the state-based model, changes

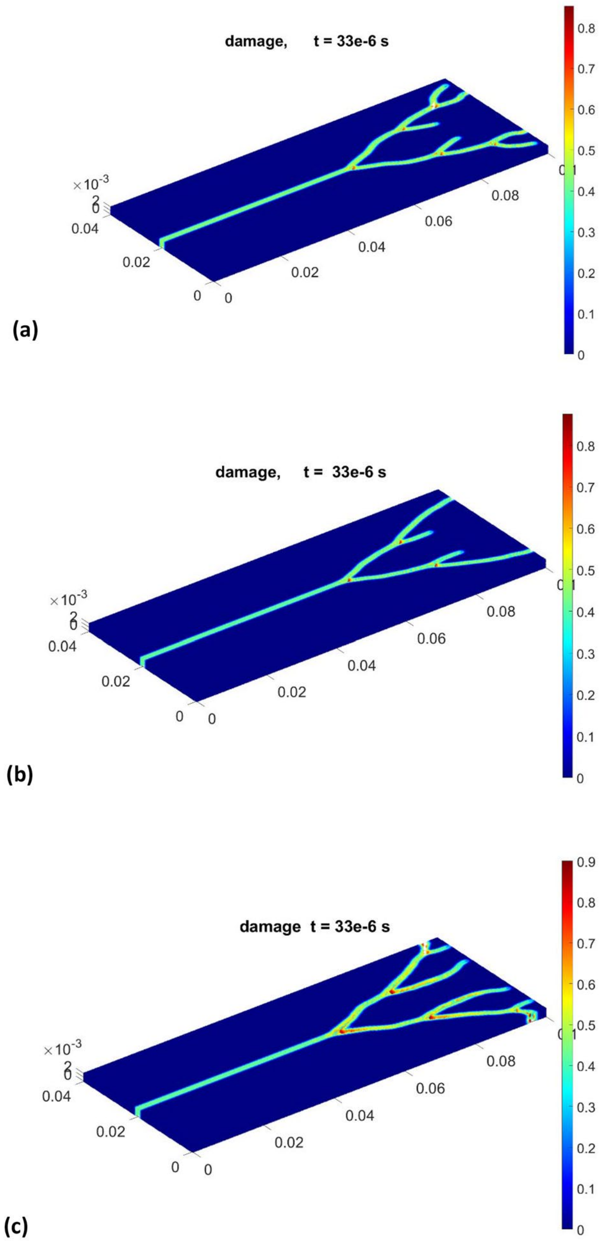


Fig. 6 Damage index profiles in glass obtained from PeriFast 3D analysis with **a** the bond-based, **b** the linearized state-based, and **c** the correspondence models

value, while in bond-based models, unless PD surface correction algorithms (e.g., see [37]) are enforced, the bulk parameters are used everywhere.

We tested a state-based model, to compare with the results from the bond-based shown in Fig. 6a, by setting $\nu = 0.25$ and $m = \pi\delta^4$ at all points in the domain (independent on whether they are near a boundary or not). We obtained results identical to the bond-based model.

While the bond-based and native state-based models differ mostly near surfaces as described above, the correspondence model is intrinsically different from the other two, making use of a "translation" between PD concepts (force and displacements maps) and classical continuum mechanics quantities (stresses and strains tensors) and employing, a local constitutive model for defining the stress–strain relationship.

As the horizon goes to zero, one expects the bond-based and native state-based models approach identical solutions since their near-the-surface differences vanish. The correspondence model, in the limit of δ-convergence, and for well-behaved problems, also converges to the classical solution of the corresponding problem. For problems with damage/fracture, this statement needs further investigation, which is outside the scope of the current work.

The 3D PD dynamic brittle fracture analyses, using a single processor, with over 2×10^6 nodes and over 660 time steps took about 1.15, 1.67, and 2.87 h to complete, with the bond-based, native state-based, and the correspondence models, respectively. When employing GPU-based calculations, the computational time is around 5 min, 6 min, and 11 min, for the three different constitutive models, respectively. Computations were performed on a Dell-Precision T7910 workstation PC, Intel(R) Xeon(R) CPU E5-2643 W v4 @3.40 GHz logical processors, and 128 GB of installed memory and NVIDIA Quadro M4000 GPU with 8 GB memory.

6 Summary and Possible Extensions of PeriFast/Dynamics

We introduced a compact Matlab-based code, PeriFast/Dynamics, which is an implementation of the Fast Convolution-Based Method (FCBM) for dynamic deformations and fracture problems in 3D. The current version of the code uses explicit time integration and offers three different options in terms of peridynamic (PD) material models: the linearized bond-based and ordinary state-based models for isotropic elastic materials, and the PD correspondence model for isotropic hyperelastic materials. Each of these comes with a model for brittle damage based on nodal strain energy density. The code is modularized with the explicit purpose to make it user-friendly and easier to adapt, modify, and extend to other problems. As long as the PD formulation for a particular problem can be setup to exhibit a convolutional structure, one can simply update/modify the MATLAB files defining the constitutive model for that particular problem. For example, elasto-plastic and ductile failure problems can easily be implemented with the structure of our code. The code could also be extended to include a pre-processor step that reads CAD-generated sample geometries and boundary conditions and automatically determines the characteristic functions that identify the domain and boundary regions in the computational box.

Because of the FCBM used to discretize the PD formulations, PeriFast/Dynamics' simulation run-times and memory requirements are independent of the number of neighbors of a node. Previous studies showed that the FCBM leads to speedups of tens to thousands compared against the traditional meshfree method, depending on the number of neighbors used.

We have briefly reviewed the PD governing equations for dynamic brittle fracture and the FCBM discretization, followed by describing the data structures used in the code. The general structure of PeriFast/Dynamics and detailed descriptions of each of the m-files contained in the code have been given. A demonstrative example of dynamic brittle fracture in glass in 3D, solved using three different constitutive models, has been provided, with step-by-step descriptions for input data and choices of outputs.

6.1 Possible Extensions

Note that the current version uses damage models with a single parameter, which can be calibrated to the critical fracture energy (material fracture toughness). These models work well in problems with pre-cracks, but when applied to problems with no pre-cracks, a higher and higher effective strength is found if one uses smaller and smaller horizon sizes (for a discussion of how to select a "proper" horizon size please see [51, 52]). For quasi-brittle fracture problems in bodies without pre-cracks we recommend using (and implementing), for example, the two-parameter bond-failure model (see [53]). Such an extension is immediate by defining *lambda* in *constitutive.m* as a non-binary variable with a gradual transition from 1 to 0, capturing a softening behavior at the microscale.

To implement ductile failure models, one can use, for example, the new PD correspondence model introduced and verified in [43, 44]. The PeriFast version presented here uses an explicit time integration scheme (velocity Verlet) and solves dynamic problems. Implicit solvers using iterative methods such as the nonlinear conjugate gradient method have been used with FCBM before (see [23]) and can be easily added to the code to perform static and quasi-static analyses.

PeriFast/Corrosion is one branch of the PeriFast suite of Matlab-based codes that implement the FCBM for PD models. The PeriFast/Corrosion branch solves corrosion damage problems (pitting corrosion, including with formation of lacy covers) and is described in [41]. By coupling the/Corrosion and/Dynamics code branches of PeriFast, one can solve, for example, stress-corrosion cracking problems like those in [5]. Because the code is fast and memory requirements are relatively low, one can solve such problems for samples at engineering-relevant scales.

Another possible extension of the code presented here is to model thermomechanical fracture and damage. Using the diffusion-type solver structure implemented in the/Corrosion branch of PeriFast, one can easily write a similar solver for transient thermal transport and couple it with the mechanics code/Dynamics to simulate thermomechanical fracture.

While not immediate, other interesting extensions may be possible: (1) fracture in heterogeneous materials (these could use, for example, the masking functions used in [41] to generate a polycrystalline microstructure); (2) impact and fragmentation (contact detection algorithms would be required for such models).

Appendix. Constitutive models included in PeriFast/Dynamics

1. *Linearized bond-based elastic material model*

This model is basically the linearized version of the micro-elastic solid (see [47]). The internal force density for this material is

$$\boldsymbol{L}(\boldsymbol{x},t)=\int_{\mathcal{H}_x}\chi_{\mathrm{B}}\chi'_{B}\mu\boldsymbol{f}\left(\boldsymbol{x},\boldsymbol{x'},t\right)dV_{x'}=\int_{\mathcal{H}_x}\chi_{\mathrm{B}}\chi'_{B}\lambda\lambda'\mathbf{C}(\xi)\boldsymbol{\eta}dV_{x'} \tag{A-1}$$

where $\boldsymbol{\xi}$ Is the bond vector, $\boldsymbol{\eta}$ is the relative displacement and $\mathbf{C}(\xi)=\alpha\omega(|\xi|)\frac{\xi\otimes\xi}{|\xi|^2}$ with $\alpha=\frac{12E}{\pi\delta^4}$ and $(|\xi|)=\frac{1}{|\xi|}$.

And the strain energy density is

$$W(\boldsymbol{x},t)=\frac{1}{2}\int_{\mathcal{H}_{x'}}\chi_{\mathrm{B}}\chi_B^{'}\lambda\lambda^{'}\boldsymbol{\eta}.\left(\frac{1}{2}\mathbf{C}(\xi)\boldsymbol{\eta}\right)dV_{x'} \tag{A-2}$$

The convolutional form of the internal force density and strain energy density for linearized bond-based models (Eqs. (A-1) and (A-2)) used in this work are [23]

$$L_i=\int_{\mathcal{H}_x}\chi_{\mathrm{B}}\chi_B^{'}\lambda\lambda^{'}\mathrm{C}_{ij}\eta_j dV_{x'}=\int_{\mathcal{H}_x}\chi_{\mathrm{B}}\chi_B^{'}\lambda\lambda^{'}\mathrm{C}_{ij}\left(u_j^{'}-u_j\right)dV_{x'}=\chi_{\mathrm{B}}\lambda\left\{\left[\mathrm{C}_{\mathrm{ij}}*(\chi_{\mathrm{B}}\lambda u_j)\right]-\left[\mathrm{C}_{\mathrm{ij}}*\chi_{\mathrm{B}}\lambda\right]u_j\right\} \tag{A-3}$$

$$W(\boldsymbol{x},t)=\frac{1}{2}\int_{\mathcal{H}_{x'}}\chi_{\mathrm{B}}\chi_B^{'}\lambda\lambda^{'}\eta_i\left(\frac{1}{2}\mathrm{C}_{\mathrm{ij}}\eta_j\right)dV_{x'}=\frac{1}{4}\chi_{\mathrm{B}}\lambda\left(\left[\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda u_iu_j\right]-2\left[\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda u_i\right]u_j+\left[\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda\right]u_iu_j\right) \tag{A-4}$$

2. *Linearized state-based elastic material model*

This model is the linearized version of the native state-based linear elastic solid (see [47]). The internal force density for this material is

$$\boldsymbol{L}(\boldsymbol{x},t)=\int_{\mathcal{H}_x}\chi_{\mathrm{B}}\chi_B^{'}\mu\boldsymbol{f}\left(\boldsymbol{x},\boldsymbol{x}^{'},t\right)dV_{x'}=\int_{\mathcal{H}_x}\chi_{\mathrm{B}}\chi_B^{'}\mu(\underline{\mathbf{T}}[\boldsymbol{x},t]\langle\xi\rangle-\underline{\mathbf{T}}\left[\boldsymbol{x}',t\right]\langle-\xi\rangle)dV_{x'} \tag{A-5}$$

where

$$\underline{\mathbf{T}}\langle\xi\rangle=\left(\frac{3k-5G}{m}\right)\omega(|\xi|)\vartheta\xi+\frac{15G}{m}\omega(|\xi|)\frac{\xi\otimes\xi}{|\xi|^2}\boldsymbol{\eta} \tag{A-6}$$

where k and G here are the bulk and shear moduli, respectively, and ϑ is the a nonlocal dilation [47].

The strain energy density for this linearized state-base material model is [47]

$$W(\boldsymbol{x},t)=\frac{1}{2}\left(\left(k-\frac{\alpha m}{9}\right)\vartheta^2+\alpha\int_{\mathcal{H}_x}\mu\omega(|\xi|)\boldsymbol{\eta}.\frac{\xi\otimes\xi}{|\xi|^2}\boldsymbol{\eta}dV_{x'}\right),\alpha=\frac{15G}{m} \tag{A-7}$$

Note: by adopting a Poisson ratio of one-quarter, the first terms on the right-hand side of Eq. (A-6) and (A-7) vanish and the linearized bond-based model presented by Eq. (A-1) and (A-2) is recovered for the points in the bulk as explained in Section 5.6.

The convolution structures for ϑ and m are derived in [23]. Let $\mathbf{C}(\xi)=30G\omega(|\xi|)\frac{\xi\otimes\xi}{|\xi|^2}$ and $\boldsymbol{a}(\xi)=\omega(|\xi|)\xi$ with $(|\xi|)=\frac{1}{|\xi|}$, the convolution structure for internal force density and strain energy density (Eqs. (A-6) and (A-7)) obtained as [23].

$$L_i=\chi_{\mathrm{B}}\lambda\left(\left(\frac{3k-5G}{m}\right)\left(-\vartheta\left[a_i*\chi_{\mathrm{B}}\lambda\right]-\left[a_i*\chi_{\mathrm{B}}\lambda\vartheta\right]\right)+\frac{1}{m}\left([\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda u_j]-[\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda]u_j\right)\right) \tag{A-8}$$

$$\begin{aligned}W(\boldsymbol{x},t)&=\frac{1}{2}\left(\left(k-\frac{\alpha m}{9}\right)\vartheta^2+\frac{1}{2m}\int_{\mathcal{H}_x}\chi_{\mathrm{B}}\chi_B^{'}\lambda\lambda^{'}\eta_i\mathrm{C}_{ij}\eta_j dV_{x'}\right)\\&=\frac{1}{2}\left(\left(k-\frac{\alpha m}{9}\right)\vartheta^2+\frac{1}{2m}\int_{\mathcal{H}_x}\chi_{\mathrm{B}}\chi_B^{'}\lambda\lambda^{'}\mathrm{C}_{ij}(u_i^{'}u_j^{'}-u_i^{'}u_j-u_iu_j^{'}+u_iu_j)dV_{x'}\right)\\&=\frac{1}{2}\left(\left(k-\frac{\alpha m}{9}\right)\vartheta^2+\frac{1}{2m}\chi_{\mathrm{B}}\lambda\left\{\left[\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda u_iu_j\right]-u_j\left[\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda u_i\right]-u_i\left[\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda u_j\right]+u_iu_j[\mathrm{C}_{ij}*\chi_{\mathrm{B}}\lambda]\right\}\right)\end{aligned} \tag{A-9}$$

The details on deriving the convolutional structure for linearized bond-based and state-based models are provided in [23].

3. *PD correspondence hyperelastic material model*

This model uses the correspondence formulation introduced in [32], and uses the classical Saint–Venant-Kirchhoff hyperelastic constitutive law. The internal force density for this material is

$$\boldsymbol{L}(\boldsymbol{x},t) = \int_{\mathcal{H}} \chi_{\mathrm{B}} \chi_B^{'} \mu \left(\underline{\mathbf{T}}\langle \xi \rangle - \underline{\mathbf{T}}'\langle -\xi \rangle \right) \mathrm{d}V_{x'} \tag{A-10}$$

where

$$\underline{\mathbf{T}}\langle \xi \rangle = \omega(|\xi|)\boldsymbol{\sigma}\left(\overline{\mathbf{F}}\right)\mathbf{K}^{-1}\xi \tag{A-11}$$

$\mathbf{K}$ is the shape tensor and $\boldsymbol{\sigma}$ is the first Piola–Kirchhoff (P-K) stress tensor which is in terms of $\overline{\mathbf{F}}$,the PD deformation gradient and defined based on the classical constitutive model that we use (for details on the correspondence formulation please see [32]). One issue encountered when using a PD correspondence model for problems with cracks is material instabilities in the form of zero energy modes. A number of solutions have been proposed to reduce/eliminate these zero energy modes. For a review of various strategies for stabilizing PD correspondence models please see [54]. In this work we use the method introduced in [33] in which a stabilizing term ($\underline{\mathbf{T}}^{\mathrm{s}}\langle \xi \rangle$) is added to the force state formulation $\underline{\mathbf{T}}^{\mathrm{c}}\langle \xi \rangle$ as follows:

$$\begin{gathered}\underline{\mathbf{T}}\langle \xi \rangle = \underline{\mathbf{T}}^{\mathrm{c}}\langle \xi \rangle + \underline{\mathbf{T}}^{\mathrm{s}}\langle \xi \rangle = \omega(|\xi|)\left(\boldsymbol{\sigma}\left(\overline{\mathbf{F}}\right)\mathbf{K}^{-1}\xi + \tfrac{GC}{\omega_0}\underline{\mathbf{z}}\langle \xi \rangle \right),\ \underline{\mathbf{z}}\langle \xi \rangle = \underline{\mathbf{Y}}\langle \xi \rangle - \overline{\mathbf{F}}\xi, \\ \omega_0 = \textstyle\int_{\mathcal{H}} \omega(|\xi|)\mathrm{d}V_{\xi},\ C = 18k/\pi\delta^5 \end{gathered} \tag{A-12}$$

For the Saint–Venant Kirchhoff model used in this study we have

$$\mathbf{S} = \lambda tr(\mathbf{E})\mathbf{I} + 2G\mathbf{E} \tag{A-13}$$

where $\mathbf{S}$ is the second P-K stress tensor and needed to be converted to the first P-K ($\boldsymbol{\sigma}$) stress tensor to be used in Eq. (A-11). $\mathbf{E}$ is the Lagrangian Green strain tensor, and λ and G are the Lamé constant and shear modulus of the material. $\mathbf{I}$ is the identity tensor.

For the PD correspondence model, we use the classical formulation to compute the strain energy density as $W(x,t) = \frac{1}{2}\mathbf{S}(\boldsymbol{x},t) : \mathbf{E}(\boldsymbol{x},t)$.

Let $\boldsymbol{a}(\xi) = \omega(|\xi|)\xi$, $\omega(|\xi|) = \frac{1}{|\xi|}$ and$\beta = \frac{GC}{\omega_0}$, then the convolutional form for the internal force density in the PD correspondence model (Eq. A-10)) is

$$\begin{aligned} L_i = -\chi_{\mathrm{B}}\lambda\Big(& \sigma_{ij}\mathrm{K}_{\mathrm{jp}}^{-1}\left[a_p * \chi_{\mathrm{B}}\lambda\right] + \left[\sigma_{ij}\mathrm{K}_{\mathrm{jp}}^{-1} * \chi_{\mathrm{B}}\lambda a_p\right] \\ & -\beta\Big(-2\left[\chi_{\mathrm{B}}\lambda * a_i\right] - 2\left[\chi_{\mathrm{B}}\lambda * \omega\right]u_i + 2\left[\chi_{\mathrm{B}}\lambda u_i * \omega\right] + \overline{\mathrm{F}}_{ij}\left[a_j * \chi_{\mathrm{B}}\lambda\right] + [\chi_{\mathrm{B}}\lambda\overline{\mathrm{F}}_{ij} * a_j]\Big)\Big) \end{aligned} \tag{A-14}$$

The detailed derivation of the convolutional form of the PD-correspondence model is given in [42] (see Section 10.3.4 there), and [43, 44].

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s42102-023-00097-6.

Acknowledgements This work has been supported by the National Science Foundation, USA under CMMI CDS&E Award No. 1953346, and by a Nebraska System Science award from the Nebraska Research Initiative.

Author Contribution F.M. and S. J. implemented and tested the code. All authors revised the code. All authors wrote the manuscript draft. F.B. obtained funding, coordinated the project, and revised the manuscript.

Funding National Science Foundation, USA under CMMI CDS&E Award No. 1953346.

Availability of Data and Materials The source code and input data used in the examples shown in the manuscript are available for free download at https://github.com/PeriFast/Code/tree/main/PeriFast_Dynamics

Declarations

Ethical Approval Not applicable.

Competing Interests The authors declare no competing interests.

References

1. PeriFast/Dynamics. https://github.com/PeriFast/Code. Accessed Dec 2022
2. Silling SA (2000) Reformulation of elasticity theory for discontinuities and long-range forces. J Mech Phys Solids 48(1):175–209. https://doi.org/10.1016/S0022-5096(99)00029-0
3. Hu W, Wang Y, Yu J, Yen C-F, Bobaru F (2013) Impact damage on a thin glass plate with a thin polycarbonate backing. Int J Impact Eng 62:152–165. https://doi.org/10.1016/j.ijimpeng.2013.07.001
4. Zhang G, Gazonas GA, Bobaru F (2018) Supershear damage propagation and sub-Rayleigh crack growth from edge-on impact: a peridynamic analysis. Int J Impact Eng 113:73–87. https://doi.org/10.1016/j.ijimpeng.2017.11.010
5. Chen Z, Jafarzadeh S, Zhao J, Bobaru F (2021) A coupled mechano-chemical peridynamic model for pit-to-crack transition in stress-corrosion cracking. J Mech Phys Solids 146:104203. https://doi.org/10.1016/j.jmps.2020.104203
6. Diehl P, Lipton R, Wick T, Tyagi M (2022) A comparative review of peridynamics and phase-field models for engineering fracture mechanics. Comput Mech 69:1259–1293. https://doi.org/10.1007/s00466-022-02147-0
7. Dahal B, Seleson P, Trageser J (2022) The evolution of the peridynamics co-authorship network. J Peridyn Nonlocal Model. https://doi.org/10.1007/s42102-022-00082-5
8. Javili A, Morasata R, Oterkus E, Oterkus S (2019) Peridynamics review. Math Mech Solids 24(11):3714–3739. https://doi.org/10.1177/1081286518803411
9. Silling SA, Askari E (2005) A meshfree method based on the peridynamic model of solid mechanics. Comput Struct 83(17–18):1526–1535. https://doi.org/10.1016/j.compstruc.2004.11.026
10. Macek RW, Silling SA (2007) Peridynamics via finite element analysis. Finite Elem Anal Des 43(15):1169–1178. https://doi.org/10.1016/j.finel.2007.08.012
11. Madenci E, Guven I (2015) The finite element method and applications in engineering using ANSYS®. Springer
12. Mehrmashhadi J, Bahadori M, Bobaru F (2020) On validating peridynamic models and a phase-field model for dynamic brittle fracture in glass. Eng Fract Mech 240:107355. https://doi.org/10.1016/j.engfracmech.2020.107355
13. Ren B, Wu CT (2018) A peridynamic model for damage prediction fiber-reinforced composite laminate. In 15th International LS-DYNA User Conference (p. 10). Michigan Detroit
14. Parks ML, Littlewood DJ, Mitchell JA, Silling SA (2012) Peridigm users' guide v1. 0.0. Sandia Report SAND2012-7800. https://doi.org/10.2172/1055619, https://www.osti.gov/servlets/purl/1055619
15. Chen H, Hu Y, Spencer BW (2016) A MOOSE-based implicit peridynamic thermomechanical model. In ASME International Mechanical Engineering Congress and Exposition (Vol. 50633, p. V009T12A072). American Society of Mechanical Engineers. https://doi.org/10.1115/IMECE2016-65552

16. Zaccariotto M, Mudric T, Tomasi D, Shojaei A, Galvanetto U (2018) Coupling of FEM meshes with Peridynamic grids. Comput Methods Appl Mech Eng 330:471–497. https://doi.org/10.1016/j.cma.2017.11.011
17. D'Elia M, Li X, Seleson P, Tian X, Yu Y (2021) A review of local-to-nonlocal coupling methods in nonlocal diffusion and nonlocal mechanics. J Peridyn Nonlocal Model 1–50. https://doi.org/10.1007/s42102-020-00038-7
18. Shojaei A, Hermann A, Cyron CJ, Seleson P, Silling SA (2022) A hybrid meshfree discretization to improve the numerical performance of peridynamic models. Comput Methods Appl Mech Eng 391:114544. https://doi.org/10.1016/j.cma.2021.114544
19. Shojaei A, Mossaiby F, Zaccariotto M, Galvanetto U (2018) An adaptive multi-grid peridynamic method for dynamic fracture analysis. Int J Mech Sci 144:600–617. https://doi.org/10.1016/j.ijmecsci.2018.06.020
20. Dipasquale D, Zaccariotto M, Galvanetto U (2014) Crack propagation with adaptive grid refinement in 2D peridynamics. Int J Fract 190(1–2):1–22. https://doi.org/10.1007/s10704-014-9970-4
21. Jafarzadeh S, Larios A, Bobaru F (2020) Efficient solutions for nonlocal diffusion problems via boundary-adapted spectral methods. J Peridyn Nonlocal Model 2:85–110. https://doi.org/10.1007/s42102-019-00026-6
22. Jafarzadeh S, Wang L, Larios A, Bobaru F (2021) A fast convolution-based method for peridynamic transient diffusion in arbitrary domains. Comput Methods Appl Mech Eng 375:113633. https://doi.org/10.1016/j.cma.2020.113633
23. Jafarzadeh S, Mousavi F, Larios A, Bobaru F (2022) A general and fast convolution-based method for peridynamics: applications to elasticity and brittle fracture. Comput Methods Appl Mech Eng 392:114666. https://doi.org/10.1016/j.cma.2022.114666
24. Lopez L, Pellegrino SF (2022) A fast-convolution based space-time Chebyshev spectral method for peridynamic models. Adv Cont Discr Mod 2022:70. https://doi.org/10.1186/s13662-022-03738-0
25. Lopez L, Pellegrino SF (2022) A nonperiodic Chebyshev spectral method avoiding penalization techniques for a class of nonlinear peridynamic models. Int J Numer Meth Eng 123(20):4859–4876. https://doi.org/10.1002/nme.7058
26. Lopez L, Pellegrino SF (2021) A spectral method with volume penalization for a nonlinear peridynamic model. Int J Numer Meth Eng 122(3):707–725. https://doi.org/10.1002/nme.6555
27. Lopez L, Pellegrino SF (2022) A space-time discretization of a nonlinear peridynamic model on a 2D lamina. Comput Math Appl 116:161–175. https://doi.org/10.1016/j.camwa.2021.07.004
28. Hu W, Ha YD, Bobaru F (2012) Peridynamic model for dynamic fracture in unidirectional fiber-reinforced composites. Comput Methods Appl Mech Eng 217:247–261. https://doi.org/10.1016/j.cma.2012.01.016
29. Silling SA, Lehoucq RB (2010) Peridynamic theory of solid mechanics. Adv Appl Mech 44:73–168. https://doi.org/10.1016/S0065-2156(10)44002-8
30. Bobaru F, Foster JT, Geubelle PH, Silling SA (2016) Handbook of peridynamic modeling. CRC Press
31. Du Q, Gunzburger M, Lehoucq RB, Zhou K (2013) Analysis of the volume-constrained peridynamic Navier equation of linear elasticity. J Elast 113(2):193–217. https://doi.org/10.1007/s10659-012-9418-x
32. Silling SA, Epton M, Weckner O, Xu J, Askari E (2007) Peridynamic states and constitutive modeling. J Elast 88(2):151–184. https://doi.org/10.1007/s10659-007-9125-1
33. Silling SA (2017) Stability of peridynamic correspondence material models and their particle discretizations. Comput Methods Appl Mech Eng 322:42–57. https://doi.org/10.1016/j.cma.2017.03.043
34. Behzadinasab M, Foster JT (2020) On the stability of the generalized, finite deformation correspondence model of peridynamics. Int J Solids Struct 182:64–76. https://doi.org/10.1016/j.ijsolstr.2019.07.030
35. Breitenfeld MS, Geubelle PH, Weckner O, Silling SA (2014) Non-ordinary state-based peridynamic analysis of stationary crack problems. Comput Methods Appl Mech Eng 272:233–250. https://doi.org/10.1016/j.cma.2014.01.002
36. Zhao J, Jafarzadeh S, Chen Z, Bobaru F (2020) An algorithm for imposing local boundary conditions in peridynamic models on arbitrary domains. engrXiv Preprints. https://doi.org/10.31224/osf.io/7z8qr
37. Le QV, Bobaru F (2018) Surface corrections for peridynamic models in elasticity and fracture. Comput Mech 61(4):499–518. https://doi.org/10.1007/s00466-017-1469-1
38. Scabbia F, Zaccariotto M, Galvanetto U (2021) A novel and effective way to impose boundary conditions and to mitigate the surface effect in state-based Peridynamics. Int J Numer Meth Eng 122(20):5773–5811. https://doi.org/10.1002/nme.6773

39. Behera D, Roy P, Anicode SVK, Madenci E, Spencer B (2022) Imposition of local boundary conditions in peridynamics without a fictitious layer and unphysical stress concentrations. Comput Methods Appl Mech Eng 393:114734. https://doi.org/10.1016/j.cma.2022.114734
40. Aksoylu B, Celiker F, Kilicer O (2019) Nonlocal operators with local boundary conditions in higher dimensions. Adv Comput Math 45(1):453–492. https://doi.org/10.1007/s10444-018-9624-6
41. Wang L, Jafarzadeh S, Mousavi F, Bobaru F (2023) PeriFast/Corrosion: a 3D pseudo-spectral peridynamic Matlab code for corrosion. J Peridyn Nonlocal Model, (in this issue)
42. Jafarzadeh S (2021) Novel and fast peridynamic models for material degradation and failure. Ph.D. dissertation. Mechanical and Materials Engineering, University of Nebraska-Lincoln
43. Mousavi F, Jafarzadeh S, Bobaru F (2023) A fast convolution-based method for peridynamic models in plasticity and ductile fracture. Under review
44. Mousavi F (2022) Novel and fast peridynamic models for large deformation and ductile failure. Ph.D. dissertation. Mechanical and Materials Engineering, University of Nebraska-Lincoln
45. Bobaru F, Ha YD (2011) Adaptive refinement and multiscale modeling in 2D peridynamics. Int J Multiscale Comput Eng 9(6):635–659. https://doi.org/10.1615/IntJMultCompEng.2011002793
46. Parks ML, Lehoucq RB, Plimpton SJ, Silling SA (2008) Implementing peridynamics within a molecular dynamics code. Comput Phys Commun 179(11):777–783. https://doi.org/10.1016/j.cpc.2008.06.011
47. Silling SA (2010) Linearized theory of peridynamic states. J Elast 99(1):85–111. https://doi.org/10.1007/s10659-009-9234-0
48. tecplot (n.d.) https://www.tecplot.com/downloads/
49. Bobaru F, Zhang G (2015) Why do cracks branch? A peridynamic investigation of dynamic brittle fracture. Int J Fract 196(1–2):59–98. https://doi.org/10.1007/s10704-015-0056-8
50. Ravi-Chandar K, Knauss WG (1984) An experimental investigation into dynamic fracture: IV. On the interaction of stress waves with propagating cracks. Int J Fract 26(3):189–200. https://doi.org/10.1007/BF01140627
51. Xu Z, Zhang G, Chen Z, Bobaru F (2018) Elastic vortices and thermally-driven cracks in brittle materials with peridynamics. Int J Fract 209(1–2):203–222. https://doi.org/10.1007/s10704-017-0256-5
52. Bobaru F, Hu W (2012) The meaning, selection, and use of the peridynamic horizon and its relation to crack branching in brittle materials. Int J Fract 176(2):215–222. https://doi.org/10.1007/s10704-012-9725-z
53. Niazi S, Chen Z, Bobaru F (2021) Crack nucleation in brittle and quasi-brittle materials: a peridynamic analysis. Theor Appl Fract Mech 112:102855. https://doi.org/10.1016/j.tafmec.2020.102855
54. Chen H, Spencer BW (2019) Peridynamic bond-associated correspondence model: stability and convergence properties. Int J Numer Meth Eng 117(6):713–727. https://doi.org/10.1002/nme.5973