Journal of Peridynamics and Nonlocal Modeling
https://doi.org/10.1007/s42102-023-00098-5

RESEARCH

PeriFast/Corrosion: A 3D Pseudospectral Peridynamic MATLAB Code for Corrosion

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

Received: 8 September 2022 / Accepted: 15 March 2023

Abstract
We introduce PeriFast/Corrosion, a MATLAB code that uses the fast convolution-based method (FCBM) for peridynamic (PD) models of corrosion damage. The FCBM uses the convolutional structure of PD equations and employs the Fast Fourier transform (FFT) to achieve a computational complexity of $O(NlogN)$. PeriFast/Corrosion has significantly lower memory allocation needs, $O(N)$, compared with, for example, the meshfree method with direct summation for PD models that requires $O(N^2)$. The PD corrosion model and the fast convolution-based method are briefly reviewed, and the detailed structure of the code is presented. The code efficiently solves 3D uniform corrosion (example for copper) and pitting corrosion (example for stainless steel) problems with multiple growing and merging pits, set in a complicated shape sample. Discussions on possible immediate extensions of the code to other corrosion damage problems are provided. PeriFast/Corrosion is a branch of PeriFast codes and is freely available on GitHub [1].

Keywords Peridynamics · Pitting corrosion · Corrosion · Fast Fourier transform (FFT) · Software · MATLAB

1 Introduction

Pitting corrosion is a type of localized corrosion that occurs in many alloys, such as aluminum alloys, stainless steel, zinc alloys, and magnesium alloys. Under certain conditions, pits may maintain stable growth [2] and lead to a rapid reduction in the durability of a structure.

Simulating pitting corrosion has been a challenge for many decades due to the complex chemomechanical interactions that are influenced by geometry, environmental conditions, and mechanical loading [3]. A different approach to modeling pitting corrosion has been introduced by Chen and Bobaru in 2015 [4] and further advanced in [5]. PD models for other types of corrosion have been introduced as well, including uniform corrosion [6],

✉ Florin Bobaru
fbobaru2@unl.edu

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

[2] Department of Civil and Environmental Engineering, The Pennsylvania State University, University Park, PA 16801, USA

pitting corrosion with lacy covers [5], crevice corrosion [7], galvanic corrosion [8], intergranular corrosion [9], and stress-dependent corrosion [6]. This peridynamic (PD) model treats the problem as dissolution and diffusion in the solid interface/electrolyte system, rather than just as a diffusion problem with a moving interface, as it has generally been addressed in the past. In this way, the PD corrosion model allows autonomous propagation of the corrosion front, with no need of tracking its location. By including corrosion damage that takes place in the layer affected by corrosion, the PD corrosion model has been coupled also with PD fracture models to simulate stress-dependent corrosion damage [6] and stress corrosion cracking [10]. Different from the classical local theory, the PD governing equations are in the form of integrodifferential equations (IDEs). In PD, a material point $\boldsymbol{x}$ is connected via peridynamic bonds to other points within its neighborhood $H_{\boldsymbol{x}}$ (see Fig. 1). The neighborhood of material point $\boldsymbol{x}$, also called the "horizon region," is centered at point $\boldsymbol{x}$ and usually is taken as a segment in 1D, a circular disk in 2D, and a sphere in 3D. The radius of the neighborhood is called the "horizon," δ, and it determines the range of "direct" interactions between material points. Because of its ability in handling the autonomous evolution of damage, the most widely used discretization for PD is the so-called meshfree method [11–14]. The method is based on a simple approximation of the PD integral operator using mid-point integration (equivalent to one-point Gaussian integration) on a uniform grid. Corrections for the partial nodal volumes covered by the horizon of an arbitrary discretization node have been used to improve the approximation (see [15]).

In general, nonlocality increases the computational cost for numerical methods used to approximate solutions to PD models. Various methods have been used to compute solutions to PD equations: the meshfree method based on the one-point Gaussian quadrature (meshfree PD) [11–14], the finite element method (FEM) [16, 17], etc. In meshfree PD, for each node, a loop is executed over all nodes inside its "family" of neighboring nodes (nodes located in $H_{\boldsymbol{x}}$, within δ from the current node) to compute the quadrature by direct summation of the terms. The computational cost of this method is $O(NM)$ where the total number of nodes in the discretized domain is N, and M is the number of neighbors in the horizon region of a node. Since the neighbor nodes need to be stored, the memory cost for the meshfree method is also high. The high computation cost in time and memory limits the space and time scale PD simulations can access. This is especially true for three-dimensional problems in which the number of neighbors needed for good resolution of damage is high.

To reduce the computational cost, a pseudospectral method has been introduced in [18–20] to take advantage of the convolutional structure present in some PD equations and use the Fast Fourier Transform (FFT) to compute the quadrature, instead of the direct summation, and achieve $O(NlogN)$ computational complexity, which beats $O(NM)$, when M is large, by a significant margin. The fast convolution-based method (FCBM) has been used for several PD formulations: diffusion, elastic deformation and fracture, and dissolution [18–21]. In addition to better scaling, FCBM does not need neighbor search and storage. In general, spectral-type methods are limited to problems on periodic domains. With FCBM, two methods have been used to solve problems on arbitrary domains: the embedded constraint method [19] and the volume penalization method [18]. Other variations of FCBM have been recently proposed in [22, 23].

Due to the nonlocality, the boundary conditions in PD need to be applied, in 3D, not over a surface, but a thick layer (a volume). There are numerous methods to enforce some given local-type boundary conditions in PD models, please see [24–26]. In this study, we use the mirror-based fictitious node method (see [24, 25]) which applies corresponding constrained values to PD nodes over the fictitious region (see Fig. 1), so that the desired local boundary conditions are satisfied.

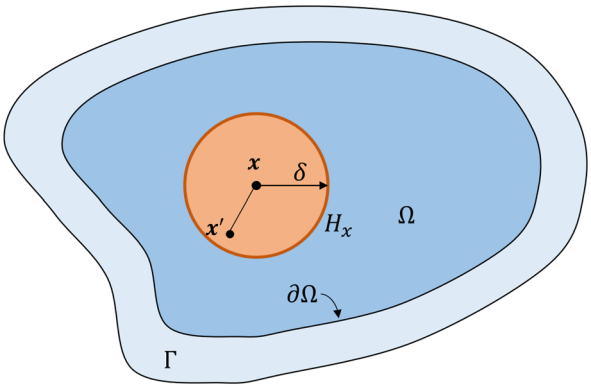

Fig. 1 A point $\boldsymbol{x}$ in a body Ω, its horizon region H_x and a neighbor $\boldsymbol{x}'$ in H_x, the boundary ($\partial\Omega$), and the (fictitious region Γ, used to enforce given local boundary conditions in the nonlocal model

In this paper, we present PeriFast/Corrosion, a MATLAB implementation of the FCBM for the PD corrosion model. The paper is organized as follows: first, we briefly review the PD corrosion model in Sect. 2; next, the FCBM discretization for PD corrosion models is described in Sect. 3; in Sect. 4, the main structure of PeriFast/Corrosion code is given, together with the GitHub link from which it can be downloaded; each code component is discussed in detail, including post-processing of the results; in Sect. 5, we demonstrate setting up input data, running the code, and visualizing the results for two corrosion examples in 3D: one on uniform corrosion and the other on pitting corrosion with salt-layer formation; users can easily reproduce the results shown here; in Sect. 6, we discuss possible extensions of the version of the code presented here to treat a variety of other corrosion types (crevice, galvanic, intergranular, etc.) and to couple with other codes when solving other corrosion-related problems (e.g., stress-corrosion cracking); concluding remarks are gathered in Sect. 7.

2 Review of the Peridynamic Model for Corrosion

In the presence of an electrolyte, the corrosion process can be modeled as a process in which the solid metal dissolves into the electrolyte, and the dissolved metal ions diffuse in the electrolyte. Consider the PD corrosion model for an arbitrary domain Ω with the fictitious region Γ:

$$\begin{cases} \frac{\partial C(\boldsymbol{x},t)}{\partial t} = L(\boldsymbol{x},t) & \boldsymbol{x}\in\Omega, t>0 \\ C(\boldsymbol{x},0) = C_0(\boldsymbol{x}) & \boldsymbol{x}\in\Omega \\ C(\boldsymbol{x},t) = g(\boldsymbol{x},t) & \boldsymbol{x}\in\Gamma, t\geq 0 \end{cases} \tag{1}$$

where $C(\boldsymbol{x},t)$ is the metal ion concentration at point $\boldsymbol{x}$ and time t, C_0 is the initial metal ion concentration, and $g(\boldsymbol{x},t)$ is a function that prescribes the constraints of concentration on the fictitious region Γ. In PeriFast/Corrosion, we use the mirror-based fictitious node method to find the profile for $g(\boldsymbol{x},t)$ such that the solution of Eq. (1) effectively matches a classical boundary condition of Dirichlet or Neumann-type. More information on how to specify $g(\boldsymbol{x},t)$ using the fictitious nodes method can be found in [19]. $L(\boldsymbol{x},t) = \int_{H_x} J\left(\boldsymbol{x},\boldsymbol{x}',t\right) dV_{\boldsymbol{x}'}$ is the summation of flow densities between point $\boldsymbol{x}$ and its neighbor points. The flow density $J\left(\boldsymbol{x},\boldsymbol{x}',t\right)$ is defined as

$$J(\boldsymbol{x},\boldsymbol{x}',t)=\begin{cases} k(\boldsymbol{x},\boldsymbol{x}',t)\frac{C(\boldsymbol{x}',t)-C(\boldsymbol{x},t)}{\|\boldsymbol{x}'-\boldsymbol{x}\|^2} & \boldsymbol{x}\ \&\ \boldsymbol{x}'\in\Omega_{\mathrm{L}} \\ q(\boldsymbol{x},\boldsymbol{x}',t)\frac{\mathrm{sign}[C(\boldsymbol{x}',t)-C(\boldsymbol{x},t)]}{\|\boldsymbol{x}'-\boldsymbol{x}\|} & (\boldsymbol{x}\in\Omega_{\mathrm{S}}\ \&\ \boldsymbol{x}'\in\Omega_{\mathrm{L}}\backslash\Omega_{\mathrm{Salt}})\text{or}\ (\boldsymbol{x}\in\Omega_{\mathrm{L}}\backslash\Omega_{\mathrm{Salt}}\ \&\ \boldsymbol{x}'\in\Omega_{\mathrm{S}}) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Here, $k(\boldsymbol{x},\boldsymbol{x}',t)$ is the micro diffusivity for the bond connecting $\boldsymbol{x}$ to $\boldsymbol{x}'$, at time t, which can be connected to the measured diffusivity of metal ions in the electrolyte by the following formulas (see [21] for details). $q(\boldsymbol{x},\boldsymbol{x}',t)$ is the micro dissolvability of the bond between $\boldsymbol{x}$ to $\boldsymbol{x}'$, at time t, which is calibrated to the corrosion current density as shown below:

$$k(\boldsymbol{x},\boldsymbol{x}',t)=\frac{9K_{\mathrm{L}}}{2\pi\delta^3},\ q(\boldsymbol{x},\boldsymbol{x}',t)=\frac{3}{\pi zF\delta^3}i \tag{3}$$

where K_{L} is the diffusivity of metal ions in the electrolyte, δ is the horizon size,i is the current density, z is the charge number, and F is Faraday's constant. Equation (3) gives the micro diffusivity and micro dissolvability for the 3D case. For 1D and 2D cases, please see [21].

During corrosion, the electrolyte can become saturated in some regions, with the concentration of dissolved metal ions reaching the saturated concentrations Csat, especially at the bottom of the corrosion pits. In this case, the ions that the electrolyte cannot sustain forming a salt layer [27]. The thickness of the salt layer increases until it balances the dissolution rate with the diffusion rate. The influence of the salt layer is discussed in the Appendix.

In Eq. (3), Ω_{L} is the liquid phase region, Ω_{S} is the dissolving solid region, and Ω_{Salt} is the salt layer region. Region Ω_{L} is defined as follows:

$$\Omega_{\mathrm{L}}=\left\{\boldsymbol{x}\in\mathbb{R}^3\,|\,\exists\, t^*\in[0,t]\ :\ C(\boldsymbol{x},t^*)<C_{\mathrm{sat}}\right\} \tag{4}$$

To locate the salt layer, we define Ω_{Salt} to identify the liquid points at saturated dissolved metal ions concentration:

$$\Omega_{\mathrm{Salt}}=\left\{\boldsymbol{x}\in\Omega_{\mathrm{L}}\,|\,C(\boldsymbol{x},t)=C_{\mathrm{sat}}\right\} \tag{5}$$

If the concentration at a liquid node reaches C_{sat}, dissolution between the solid nodes directly connected by PD bonds to this liquid node is temporarily stopped. In practice, since we are comparing the floating point numbers, not integers, the salt layer is implemented as liquid nodes whose concentrations are higher than or equal to C_{sat}.

During pitting corrosion, regions near the surface may passivate due to low pH values while corrosion continues in other regions. The corrosion process bypasses the passivated part and may create (in stainless steel, for example) a perforated cover (the "lacy cover") when it reaches back to the surface (see [5]). In the current version of the PeriFast/Corrosion code, passivation and lacy cover formation mechanisms are not included. Instead, when modeling pitting corrosion, we assume the surface (except for the initial pits) is passivated; therefore, corrosion only occurs between solid and liquid inside the pits. To locate the liquid points inside the pits, we denote the non-pitted initial solid geometry as Ω_{G}. Nodes in Ω_{G} that are in the liquid phase of the pitted sample form the $\Omega_{\mathrm{L\_pit}}$ set, defined as the intersection between Ω_{L} and Ω_{G}. Similarly, we define the salt layer inside pits. More details on the implementation of uniform and pitting corrosion are presented in the next section.

$$\begin{aligned} \Omega_{\text{L\_pit}} &= \left\{ \boldsymbol{x} \in \Omega_{\text{L}} \cap \Omega_{\text{G}} \right\} \\ \Omega_{\text{Salt\_pit}} &= \left\{ \boldsymbol{x} \in \Omega_{\text{Salt}} \cap \Omega_{\text{G}} \right\} \end{aligned} \tag{6}$$

The dissolving solid domain Ω_{S} is defined as

$$\Omega_{\text{S}} = \left\{ \boldsymbol{x} \in \Omega \backslash \Omega_{\text{L}} \right\} \tag{7}$$

The model above, discretized using the meshfree method, has been validated against various experimental results on uniform corrosion [6], pitting corrosion [5], crevice corrosion [7], intergranular corrosion [9], and galvanic corrosion [8]. Verification of the numerical implementation against analytical classical solution has been presented for galvanic corrosion in [8].

The FCBM solution for the above model has been introduced in [21], where results have been also verified against classical solutions and validated against experiments. To account for material variability, randomness has been introduced in the meshfree implementation at the bond level. The bonds are randomly selected to be damaged to match the corrosion damage value at each node (see [4]). In FCBM, a different way of introducing the expected randomness in the material has to be used. In [21], this step is achieved by defining the micro dissolvability q as a function that depends on location via a stochastic characteristic function.

3 Brief Review of the Fast Convolution-Based Method for PD Models of Corrosion

PeriFast/Corrosion uses the fast convolution-based method (FCBM) to solve the corrosion problem in Eq. (1). The fast convolution-based method (FCBM) assumes a given domain of arbitrary shape $\Omega \cup \Gamma$ is embedded in a "tight" box $\mathbb{T}$, which is then extended by periodicity in all Cartesian directions, as shown in Fig. 2. Note that if the fictitious region (here of thickness δ) is not used for the imposition of boundary conditions, one has to leave a space δ between the domain and the edge of the box to avoid a

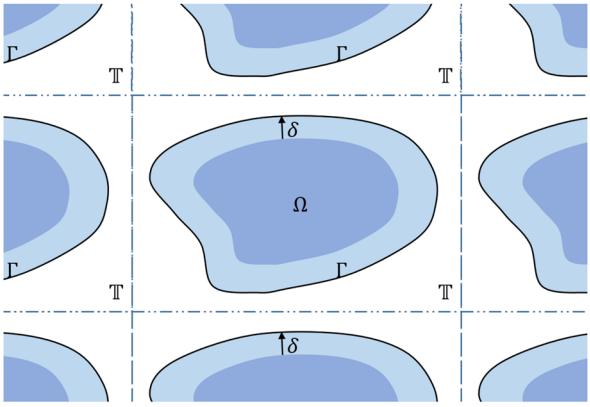

Fig. 2 Extension of a peridynamic body to a periodic box

wrap-around effect when constructing the periodic solution. To be able to treat arbitrary boundary conditions, the "embedded constraint" approach [19] can be used.

After extending the problem domain, we replace Eq. (1) with the following periodic problem:

$$\begin{cases} \frac{\partial C(\boldsymbol{x},t)}{\partial t} = \chi_{\Omega}(\boldsymbol{x})[L(\boldsymbol{x},t)] + \chi_{\Gamma\cup\Lambda}(\boldsymbol{x})\frac{\partial C_w(\boldsymbol{x},t)}{\partial t} & \text{for } \boldsymbol{x} \in \mathbb{T}, t > 0 \\ C(\boldsymbol{x},0) = C_0 \quad \text{for } \boldsymbol{x} \in \mathbb{T} \end{cases} \tag{8}$$

where $\Lambda = \mathbb{T}\backslash(\Omega \cup \Gamma)$ is the gap region, χ_{Ω} and $\chi_{\Gamma\cup\Lambda}$ are characteristic functions used to distinguish between the subdomains, and C_w is a function that is equal to the volume constraints on Γ and zero elsewhere:

$$\begin{aligned} \chi_{\Omega}(\boldsymbol{x}) &= \begin{cases} 1 & \boldsymbol{x} \in \Omega \\ 0 & \boldsymbol{x} \in \mathbb{T}\backslash\Omega = \Gamma \cup \Lambda \end{cases} \\ \chi_{\Gamma\cup\Lambda}(\boldsymbol{x}) = 1 - \chi_{\Omega}(\boldsymbol{x}) &= \begin{cases} 0 & \boldsymbol{x} \in \Omega \\ 1 & \boldsymbol{x} \in \mathbb{T}\backslash\Omega = \Gamma \cup \Lambda \end{cases} \\ C_w(\boldsymbol{x},t) &= \begin{cases} 0 & \boldsymbol{x} \in \Omega \cup \Lambda \\ C_{\Gamma}(\boldsymbol{x},t) & \boldsymbol{x} \in \Gamma \end{cases} \end{aligned} \tag{9}$$

where $C_{\Gamma}(\boldsymbol{x},t)$ is the volume constraint calculated by the mirror-based fictitious node method from the given local boundary conditions (Dirichlet- or Neumann-type). Notice that the solution to Eq. (8) on Ω is the same as the solution to Eq. (1). Since Eq. (8) is defined over a periodic domain, we can use the FFT for fast computation of the circular convolutions arising from the equation.

To apply the FCBM to PD formulations, these have to be set up in a convolutional form. First, we define the following characteristic functions to identify the phases in Ω:

$$\begin{aligned} \chi_{\mathrm{L}}(\boldsymbol{x},t) &= \begin{cases} 1 & \boldsymbol{x} \in \Omega_{\mathrm{L}} \\ 0 & \text{else} \end{cases} \\ \chi_{\mathrm{Salt}}(\boldsymbol{x},t) &= \begin{cases} 1 & \boldsymbol{x} \in \Omega_{\mathrm{Salt}} \\ 0 & \text{else} \end{cases} \\ \chi_{\mathrm{S}}(\boldsymbol{x},t) &= 1 - \chi_{\mathrm{L}}(\boldsymbol{x},t) \end{aligned} \tag{10}$$

where χ_{L} represents the liquid subdomain, χ_{Salt} is the salt layer subdomain, and χ_{S} represents the solid subdomain.

Since $\chi_{\mathrm{L}}(\boldsymbol{x},t) + \chi_{\mathrm{S}}(\boldsymbol{x},t) = 1$, we can write:

$$\begin{aligned} &\int_{H_x} J(\boldsymbol{x},\boldsymbol{x}',t)\mathrm{d}\boldsymbol{x}' = \\ &\left[\chi_{\mathrm{L}}(\boldsymbol{x},t) + \chi_{\mathrm{S}}(\boldsymbol{x},t)\right]\int_{H_x}\left[\chi_{\mathrm{L}}(\boldsymbol{x}',t) + \chi_{\mathrm{S}}(\boldsymbol{x}',t)\right]J(\boldsymbol{x},\boldsymbol{x}',t)\mathrm{d}\boldsymbol{x}' = \\ &\chi_{\mathrm{L}}\int_{H_x}\chi'_{\mathrm{L}}\,J\,\mathrm{d}\boldsymbol{x}' + (\chi_{\mathrm{L}} - \chi_{\mathrm{Salt}})\int_{H_x}\chi'_{\mathrm{S}}\,J\,\mathrm{d}\boldsymbol{x}' + \chi_{\mathrm{S}}\int_{H_x}(\chi'_{\mathrm{L}} - \chi'_{\mathrm{Salt}})\,J\,\mathrm{d}\boldsymbol{x}' \end{aligned} \tag{11}$$

where χ_{L} is short for $\chi_{\mathrm{L}}(\boldsymbol{x},t)$ and χ'_{L} is short for $\chi_{\mathrm{L}}(\boldsymbol{x}',t)$ and similar for χ_{Salt} and χ_{S}.

The splitting of the original integral into four integrals means that each new integral represents metal ion transfer via bonds connecting different phases. For example, $\chi_{\mathrm{S}}\int_{H_x}(\chi'_{\mathrm{L}}-\chi'_{\mathrm{Salt}})\,J\,\mathrm{d}\boldsymbol{x}'$ represents the dissolution between solid nodes and liquid nodes whose concentration is below C_{sat}. Since ion transport inside the solid is minute compared with the liquid–solid or liquid–liquid transport, we take the transport between solid nodes to be zero, meaning that only three terms are left in Eq. (11).

In uniform corrosion, Eq. (11) is applied to all points near the surface. For the pitting corrosion model, the surface is considered passivated except for the initial pits, and, as discussed in the previous section, corrosion/dissolution involves the solid points near the pit surfaces and the electrolyte inside the pits. To determine these regions (liquid-phase nodes inside pits), we define the following characteristic functions:

$$\chi_{\mathrm{L\_pit}}(\boldsymbol{x},t)=\begin{cases}1 & \boldsymbol{x}\in\Omega_{\mathrm{L}}\cap\Omega_{\mathrm{G}}\\ 0 & \text{else}\end{cases}$$

$$\chi_{\mathrm{Salt\_pit}}(\boldsymbol{x},t)=\begin{cases}1 & \boldsymbol{x}\in\Omega_{\mathrm{Salt}}\cap\Omega_{\mathrm{G}}\\ 0 & \text{else}\end{cases} \tag{12}$$

By replacing the $(\chi_{\mathrm{L}}-\chi_{\mathrm{Salt}})$ with $(\chi_{\mathrm{L\_pit}}-\chi_{\mathrm{Salt\_pit}})$ and changing $(\chi'_{\mathrm{L}}-\chi'_{\mathrm{Salt}})$ to $(\chi'_{\mathrm{L\_pit}}-\chi'_{\mathrm{Salt\_pit}})$ in Eq. (11), we have

$$\begin{aligned}\int_{H_x}J(\boldsymbol{x},\boldsymbol{x}',t)\,\mathrm{d}\boldsymbol{x}' =&\chi_{\mathrm{L}}\int_{H_x}\chi'_{\mathrm{L}}J\,\mathrm{d}\boldsymbol{x}'\\ &+\left(\chi_{\mathrm{L\_pit}}-\chi_{\mathrm{Salt\_pit}}\right)\int_{H_x}\chi'_{\mathrm{S}}J\,\mathrm{d}\boldsymbol{x}'\\ &+\chi_{\mathrm{S}}\int_{H_x}\left(\chi'_{\mathrm{L\_pit}}-\chi'_{\mathrm{Salt\_pit}}\right)J\,\mathrm{d}\boldsymbol{x}'\end{aligned} \tag{13}$$

By entering the flow density J defined in Eq. (2) into Eq. (11) and Eq. (13), we obtain the convolution form needed, for each of the two cases discussed:

Uniform corrosion:

$$\begin{aligned}\int_{H_x}J\,\mathrm{d}\boldsymbol{x}' =&\chi_{\mathrm{L}}\int_{H_x}\chi'_{\mathrm{L}}\omega_{\mathrm{diff}}\left(\|\boldsymbol{x}'-\boldsymbol{x}\|\right)C'\,\mathrm{d}\boldsymbol{x}'-\chi_{\mathrm{L}}C\int_{H_x}\chi'_{\mathrm{L}}\omega_{\mathrm{diff}}\left(\|\boldsymbol{x}'-\boldsymbol{x}\|\right)\mathrm{d}\boldsymbol{x}'\\ &+(\chi_{\mathrm{L}}-\chi_{\mathrm{Salt}})\int_{H_x}\chi'_{\mathrm{S}}\omega_{\mathrm{corr}}\left(\|\boldsymbol{x}'-\boldsymbol{x}\|\right)\mathrm{d}\boldsymbol{x}'-\chi_{\mathrm{S}}\int_{H_x}(\chi'_{\mathrm{L}}-\chi'_{\mathrm{Salt}})\omega_{\mathrm{corr}}\left(\|\boldsymbol{x}'-\boldsymbol{x}\|\right)d\boldsymbol{x}'\\ =&\chi_{\mathrm{L}}\left[\omega_{\mathrm{diff}}*\left(\chi_{\mathrm{L}}C\right)\right]-\left(\chi_{\mathrm{L}}C\right)\left[\omega_{\mathrm{diff}}*\chi_{\mathrm{L}}\right]\\ &+(\chi_{\mathrm{L}}-\chi_{\mathrm{Salt}})\left[\omega_{\mathrm{corr}}*\chi_{\mathrm{S}}\right]-\chi_{\mathrm{S}}[\omega_{\mathrm{corr}}*(\chi_{\mathrm{L}}-\chi_{\mathrm{Salt}})]\end{aligned}$$

Pitting corrosion:

$$\begin{aligned}\int_{H_x}J\,\mathrm{d}\boldsymbol{x}' =&\chi_{\mathrm{L}}\left[\omega_{\mathrm{diff}}*\left(\chi_{\mathrm{L}}C\right)\right]-\left(\chi_{\mathrm{L}}C\right)\left[\omega_{\mathrm{diff}}*\chi_{\mathrm{L}}\right]\\ &+(\chi_{\mathrm{L\_pit}}-\chi_{\mathrm{Salt\_pit}})\left[\omega_{\mathrm{corr}}*\chi_{\mathrm{S}}\right]\\ &-\chi_{\mathrm{S}}[\omega_{\mathrm{corr}}*(\chi_{\mathrm{L\_pit}}-\chi_{\mathrm{Salt\_pit}})]\end{aligned} \tag{14}$$

where $\omega_{\text{diff}}\left(\|\boldsymbol{x}'-\boldsymbol{x}\|\right)=\frac{k}{\|\boldsymbol{x}'-\boldsymbol{x}\|^2}$ and $\omega_{\text{corr}}\left(\|\boldsymbol{x}'-\boldsymbol{x}\|\right)=\frac{q}{\|\boldsymbol{x}'-\boldsymbol{x}\|}$. See extensions to other types of corrosion damage; please see the discussion in Sect. 6.

To spatially discretize the periodic domain $\mathbb{T}$, we use a uniform grid [18]:

$$\boldsymbol{x}_{nmp}=\left(n\Delta x_1, m\Delta x_2, p\Delta x_3\right), \text{with } \Delta x_1=\frac{L_1}{N_1};\Delta x_2=\frac{L_2}{N_2};\Delta x_3=\frac{L_3}{N_3} \tag{15}$$

where N_1, N_2, N_3 are the number of nodes in each direction, $n=\{0,\ldots,N_1-1\};m=\{0,\ldots,N_2-1\};p=\{0,\ldots,N_3-1\}$, and L_1, L_2, L_3 are the dimensions of $\mathbb{T}$.

In the Fourier space, the circular convolution is transformed into a multiplication of the convoluted functions. By denoting $\mathbf{F}$ and $\mathbf{F}^{-1}$ the FFT and inverse FFT [18, 19], to numerically calculate the $\omega_{\text{diff}} * \left(\chi_{\text{L}}C\right)$ in Eq. (14), we take the Fourier Transform of ω^s_{diff} and $\chi_{\text{L}}C$, multiply them and take their inverse Fourier Transform $\mathbf{F}^{-1}[\mathbf{F}\left(\omega^s_{\text{diff}}\right)\mathbf{F}(\chi_{\text{L}}C)]$. Notice that the ω^s_{diff} is the shifted kernel with respect to the box coordinates. This shift is necessary for the circular convolution operation. The figure below shows the original version and the shifted version of a 2D radial kernel. The colored disk represents the non-zero part of the kernel Fig. 3.

The concentration at each time step is updated using the forward Euler time-integration algorithm, and for each of the cases discussed, we have:

Uniform corrosion:

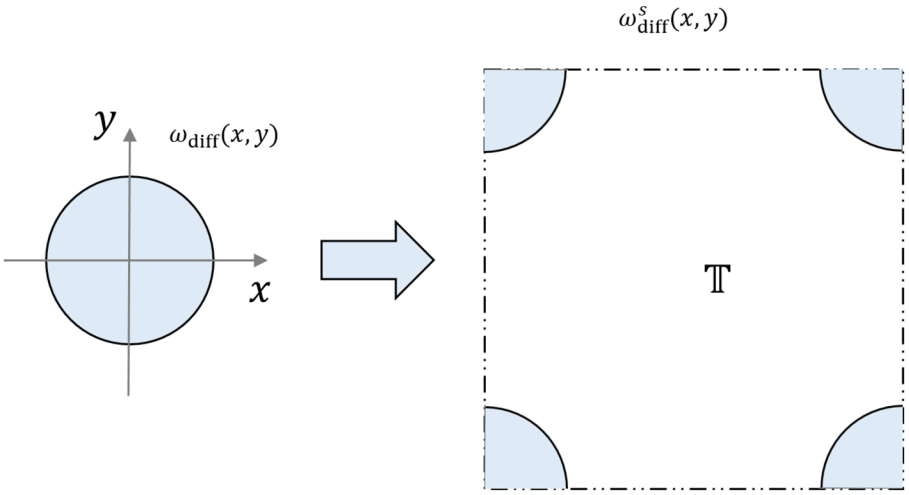

Fig. 3 The support of the kernel function (ω_{diff}) in its original form (left) and its shifted form in the box $\mathbb{T}$ (right)

$$
\begin{aligned}
C^{nmp,t+\Delta t} = & \chi_{\Omega}^{nmp}\Big(\chi_{\mathrm{L}}^{nmp,t}\mathbf{F}^{-1}\left[\mathbf{F}\left(\omega_{\mathrm{diff}}^{s}\right)\mathbf{F}\left(\chi_{\mathrm{L}}C\right)\right]\Big|^{nmp,t}\Delta V \\
& -\left(\chi_{\mathrm{L}}^{nmp,t}C^{nmp,t}\right)\mathbf{F}^{-1}\left[\mathbf{F}\left(\omega_{\mathrm{diff}}^{s}\right)\mathbf{F}\left(\chi_{\mathrm{L}}\right)\right]\Big|^{nmp,t}\Delta V \\
& +\left(\chi_{\mathrm{L}}^{nmp,t}-\chi_{\mathrm{Salt}}^{nmp,t}\right)\mathbf{F}^{-1}\left[\mathbf{F}\left(\omega_{\mathrm{corr}}^{s}\right)\mathbf{F}\left(\chi_{\mathrm{S}}\right)\right]\Big|^{nmp,t}\Delta V \\
& -\chi_{\mathrm{S}}^{nmp,t}\mathbf{F}^{-1}\left[\mathbf{F}\left(\omega_{\mathrm{corr}}^{s}\right)\mathbf{F}\left(\chi_{\mathrm{L}}-\chi_{\mathrm{Salt}}\right)\right]\Big|^{nmp,t}\Delta V\Big) \\
& +\left(1-\chi_{\Omega}^{nmp}\right)C_{w}^{nmp,t}
\end{aligned}
$$

Pitting corrosion:

$$
\begin{aligned}
C^{nmp,t+\Delta t} = & \chi_{\Omega}^{nmp}\Big(\chi_{\mathrm{L}}^{nmp,t}\mathbf{F}^{-1}\left[\mathbf{F}\left(\omega_{\mathrm{diff}}^{s}\right)\mathbf{F}\left(\chi_{\mathrm{L}}C\right)\right]\Big|^{nmp,t}\Delta V \\
& -\left(\chi_{\mathrm{L}}^{nmp,t}C^{nmp,t}\right)\mathbf{F}^{-1}\left[\mathbf{F}\left(\omega_{\mathrm{diff}}^{s}\right)\mathbf{F}\left(\chi_{\mathrm{L}}\right)\right]\Big|^{nmp,t}\Delta V \\
& +\left(\chi_{\mathrm{L\_pit}}^{nmp,t}-\chi_{\mathrm{Salt\_pit}}^{nmp,t}\right)\mathbf{F}^{-1}\left[\mathbf{F}\left(\omega_{\mathrm{corr}}^{s}\right)\mathbf{F}\left(\chi_{\mathrm{S}}\right)\right]\Big|^{nmp,t}\Delta V \\
& -\chi_{\mathrm{S}}^{nmp,t}\mathbf{F}^{-1}\left[\mathbf{F}\left(\omega_{\mathrm{corr}}^{s}\right)\mathbf{F}\left(\chi_{\mathrm{L\_pit}}-\chi_{\mathrm{Salt\_pit}}\right)\right]\Big|^{nmp,t}\Delta V\Big) \\
& +\left(1-\chi_{\Omega}^{nmp}\right)C_{w}^{nmp,t}
\end{aligned}
\tag{16}
$$

where $\Delta V = \Delta x_1 \Delta x_2 \Delta x_3$ and the argument $(\boldsymbol{x}_{nmp}, t)$ is replaced with superscript nmp, t to reduce the notational complexity.

Using the FFT to numerically calculate $\mathbf{F}$ and $\mathbf{F}^{-1}$ at each time step, the computational complexity is reduced to $O(NlogN)$. Since there is no need to store the neighbor nodes, the memory cost is also reduced from $O(N^2)$ to $O(N)$. The efficiency gains compared with the direct summation for the meshfree method quadrature have been discussed in [18, 19, 21].

4 The PeriFast/Corrosion Code

In this section, we first introduce the overall structure of PeriFast/Corrosion code and then discuss in detail each of the m-files contained in it. The PeriFast/Corrosion code can be found at https://github.com/PeriFast/Code. The user could download the code by clicking on the "Code" and then "Download ZIP," as shown in Fig. 3. If the user has Git Bash installed, the code can be downloaded by typing "*git clone* https://github.com/PeriFast/Code" in the Git Bash command window. The download will contain the PeriFast/Dynamics code, which solves dynamic brittle fracture problems (see [28]).

In the current version, the code can handle two different types of corrosion: uniform corrosion and pitting corrosion starting from a set of predefined initial pits. The user can easily switch between these two options by changing the *corrosion_type* in *input.m*.

PeriFast/Corrosion consists of thirteen MATLAB (.m) files, including *main.m*, *inputs.m*, *nodes_and_sets.m*, *initial_conditions.m*, *kernel_functions.m*, *boundary_conditions.m*, *initial_gpu_arrays.m*, *update_VC.m*, *update_C.m*, *dump_output.m*, *dump_output_Tecplot.m*, *visualization.m*, and *postprocess.m*.

The *main.m* file is the main script file to run PeriFast/Corrosion. It calls all the scripts and functions required to run the corrosion simulation. *inputs.m* is the input script that contains all the physical parameters. *nodes_and_sets.m* describes the domain size, geometry, discretized nodes coordinates, horizon size, and the discrete characteristic functions for subdomains. Currently, the user needs to manually setup the geometry data for each example. Readers can contribute to the code by adding functions that would automatically use geometry data exported from various CAD systems. *initial_conditions.m* is the file that defines the initial concentration and the locations and sizes of initial pits. *kernel_functions.m* specifies the kernel functions used in the corrosion model. *boundary_conditions.m* defines the type and values of boundary conditions. *initial_gpu_arrays.m* copies the arrays to GPU memory if the user chooses to use GPU to speed up the computation. *update_VC.m* is used to update the fictitious nodes' concentration, based on the boundary conditions and concentration at each time step. *update_C.m* is used to update the concentration and characteristic functions at each time step. *dump_output.m* and *dump_output_Tecplot.m* are called, at the frequency defined by the user, to separately store the output data in a MATLAB variable and a Tecplot file. *visualization.m* is the script that plots data at certain times specified in *inputs.m*. *postprocess.m* is the script that the user could use to visualize the output data.

The visualization part affects the runtime. When running speed tests or large simulations, we suggest using the visualization part as a postprocessing step, by turning it off. The user can turn off visualization by setting the variable *is_plot_in_Matlab* equal to zero in *inputs.m* file. In such a case, the user needs to use the MATLAB output data file (*Results.mat*) and *postprocess.m* to postprocess the results.

4.1 Description of Main.m

This file consists of the scripts needed to run the program. In MATLAB, since Released 2008a, the multithreaded computations have been on by default. MATLAB will determine the most desirable number of threads. The structure is shown in Algorithm 1.

Algorithm 1 Structure of main.m

Call *inputs.m* to get the input parameters (see section 4.2)

Call *nodes_and_sets.m* to discretize the domain (see section 4.3)

Call *initial_conditions.m* to assign initial conditions and initial pits (see section 4.4)

Call *kernel_functions.m* to define and shift the kernel functions (see section 4.5)

Call *boundary_conditions.m* to define the boundary conditions (see section 4.6)

If the user chooses to output data to Tecplot file

    Create a structure array *tdata* to store the output data (node coordinates and concentration)

If the user chooses to visualize during runtime and create an animation

    Create a VideoWriter object to capture the plotted figures as frames in a video file (.avi format)

If the user chooses to run on GPU

    Call *initial_gpu_arryas.m* to transform data to GPU

For each time step

    Call *update_VC.m* to update the volume constraints (see section 4.6)

    Call *update_C.m* to update node concentration using the forward Euler method (see Eq. (16)) and update characteristic functions (based on definitions in Eq. (10) and Eq. (12))

    At a certain time, dump output data and visualize the results (see section 4.9) if the user chooses to do so in *inputs.m*

End

If the user chooses to visualize during runtime and create an animation

    Close the VideoWriter object

If the user chooses to output data to Tecplot file

    Write the structure array *tdata* to a .plt file, using an opensource MATLAB script

    (https://github.com/wme7/Aero-matlab/blob/master/Tecplot/mat2tecplot.m)

Save the output data to a MAT file.

4.2 Description of Inputs.m

This file consists of all the input physical parameters for the corrosion simulation, as well as data needed to define outputs and their frequency. To run a corrosion simulation, the user needs to define the following physical parameters and output-related data. We also introduce functions to check all these input parameters and display error/warning messages when input data is out of the expected range. For example, if the *corrosion_type* is set to values other than 0 or 1, an error message will remind user to change it. The details of messages can be found in *inputs.m*.

Algorithm 2 Structure of inputs.m

Corrosion type (*corrosion_type*), 0 represents uniform corrosion and 1 represents pitting corrosion
Diffusion coefficient in the electrolyte (K), unit in m^2/s
The current density of corrosion (i), unit in A/m^2
Average charge number of material (n)
Dissolution flux $q = i/nF$, F is Faraday's constant ($96485.33\ \mathrm{C}/\mathrm{mol}$)
The saturated concentration of dissolved metal ions in electrolyte (C_{sat}), unit in $\mathrm{mol}/\mathrm{m}^3$
The concentration of metal atoms in the solid (C_{solid}), unit in $\mathrm{mol}/\mathrm{m}^3$
Simulation time (t_{max}), unit in seconds
Time step (dt), unit in seconds
Boolean value (*is_plot_in_Matlab*), whether to plot and output the animation
Boolean value (*is_output_to_Tecplot*), whether to write data to .plt file
Time interval of plotting and dumping output data (*t_output*), unit in seconds
Boolean value (*has_salt_layer*), whether to consider salt layer effect
Boolean value (*run_in_gpu*), whether to use GPU for computation
Function *input_check* which checks all above inputs and displays error/warning messages

4.3 Description of Nodes_and_sets.m

In this file, the user needs to define the geometrical information of the problem. As shown in Fig. 2, given a PD body Ω, the user will first define a rectangular box $\mathbb{T}$ whose edges are aligned with the coordinate system. The PD body Ω and its fictitious region Γ should be enclosed in the rectangular box $\mathbb{T}$ as tight as possible; this keeps the number of excess nodes to a minimum. Since now the edge of $\mathbb{T}$ is at least δ away from the PD body Ω, there will be no "wrap-around" effect. After discretization, the nodal coordinates are stored as $N_y \times N_x \times N_z$ matrices. The solid geometry is represented as a characteristic function, which is 1 for nodes located inside the geometry and 0 otherwise. In the numerical example shown, the coordinate system is chosen in the center of the box. A function checks the parameters defined in this script and displays error/warning messages when values could create problems. For example, if the m-factor value is less than three, a warning message will appear to remind the user to set a finer grid. More details of messages can be found in *nodes_and_sets.m*.

Algorithm 3 Structure of Nodes_and_sets.m

Define the horizon size (δ), unit in m

Define the physical domain size (L_x, L_y, L_z), unit in m

Extend the physical domain to a period box and define the size (B_x, B_y, B_z), unit in m

Define the number of nodes in each direction (N_x, N_y, N_z)

Calculate grid size in each direction (dx, dy, dz), unit in m

Create a uniform grid, nodal coordinates stored in X, Y, Z

Calculate *m*-factor value

Build the characteristic functions for the PD body (χ) and the initial solid geometry (χ_N)

Function *discretization_check* which looks over all the above parameters and displays error/warning messages

4.4 Description of Initial_conditions.m

The user needs to define the initial conditions of the corrosion simulation in this file. Notice that in the pitting corrosion example shown, the initial pits' location and size are randomly generated on a single surface, using a uniform probability distribution. The user can specify the total initial pits number and the probability distribution for the location of centers of spheres and their radii used to produce the initial pits by intersecting these spheres with the solid sample. These intersections lead to initial pits having spherical cap shapes. The region in which the centers of spheres are defined is currently selected to be slightly above the corroding surface (up to 2 μm), and the radii are selected inside the [2, 4] μm interval.

Algorithm 4 Structure of initial_conditions.m

Create an initial concentration matrix C_0 and assign C_{solid} to the solid part

If the corrosion type is pitting corrosion (*corrosion_type* equals to 1)

    Define the total number of initial spherical shape pits

    For each initial pit

        Randomly generate pit center coordinates, with uniform distribution

        Randomly generate pit radius, with uniform distribution

        Assign initial concentration to the nodes inside the pit

    End

End

Initialize the characteristic functions for the liquid region (χ_{L}), the liquid region in pits ($\chi_{\mathrm{L\_pit}}$), the salt layer (χ_{Salt}), and the solid region (χ_{S}) based on the definition in Eq. (10)

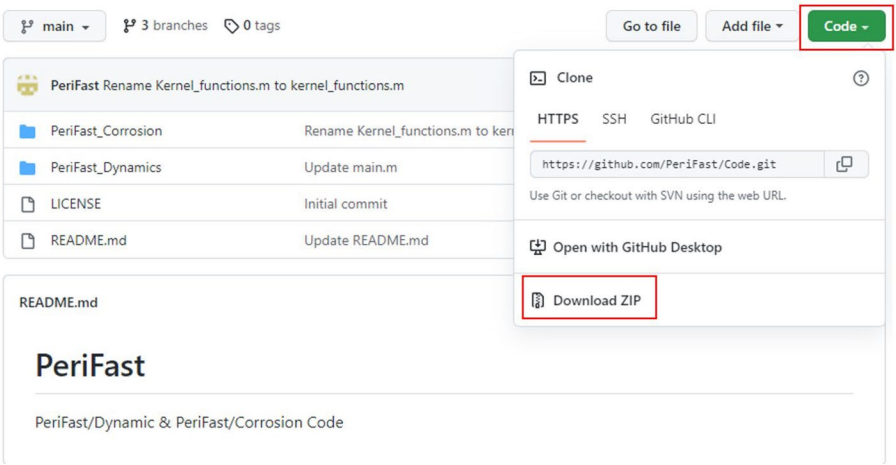

Fig. 4 PeriFast Github download process

4.5 Description of Kernel_functions.m

In this file, the kernel functions (ω_{diff} and ω_{corr}) used in Eq. (14) are defined and discretized. Notice that the kernel functions are shifted such that the origins coincide with the corners of the periodic box $\mathbb{T}$, described in the previous section (see Fig. 4). The *fftshift* function in MATLAB is used to shift the kernel functions. This function cuts the array at the mid-planes of the box, swaps the subdivisions, and returns the desired shifted function. More detailed information about the *fftshift* function can be found in MATLAB documentation.

Based on the stability analysis in [18, 19], we write a function *stability_check* that can display a warning message if the time step might be too large relative to the horizon size and the diffusion parameters used.

Algorithm 5 Structure of Kernel_functions.m

Define kernel functions for diffusion ω_{diff}, corrosion ω_{corr}

Discretize these kernel functions $\omega_{\text{diff}}^{nmp} = \omega_{\text{diff}}(\boldsymbol{x}_{nmp})$, $\omega_{\text{corr}}^{nmp} = \omega_{\text{corr}}(\boldsymbol{x}_{nmp})$

Translate these three kernel functions such that their origin coincides with the box center, and then shift them using *fftshift*: $\omega^{s}{}_{\text{diff}}^{nmp} = fftshift\left(\omega_{\text{diff}}(x_n - x_c, y_m - y_c, z_p - z_c)\right)$, $\omega^{s}{}_{\text{corr}}^{nmp} = fftshift(\omega_{\text{corr}}(x_n - x_c, y_m - y_c, z_p - z_c))$, where (x_c, y_c, z_c) is the center of the periodic box $\mathbb{T}$

Compute FFT of the shifted kernel functions $\omega^{s}{}_{\text{diff}}^{nmp}$ $\omega^{s}{}_{\text{corr}}^{nmp}$

Function *stability_check* to check whether the time step satisfies the stability condition [19][21], and display warning messages

4.6 Description of Boundary_Conditions.m and Update_VC.m

In boundary_conditions.m, the user defines the boundary conditions type and their value. The variable BC_type is 1 for the Dirichlet boundary condition and 2 for the Neumann boundary condition. In update_VC.m, the fictitious nodes method is implemented to apply/update the corresponding volume constraints to the nodes in Γ. In the fictitious node method, the volume constraint values are calculated based on the given local boundary condition and the solution values on the interior side of the boundary. More details of the mirror-based fictitious nodes method can be found in Appendix A in [15].

Algorithm 6 Structure of Boundary_conditions.m & update_VC.m

boundary_conditions.m:

Define the boundary conditions type (Dirichlet or Neuman) and its value

Locate the nodes on the interior side of the boundary

update_VC.m:

Use fictitious node method to update C_w based on the boundary conditions and C at nodes located at the interior side of the given domain boundary

Function *boundary_condition_check* verifies the boundary conditions types and returns error messages if input data has issues.

4.7 Description of Initial_gpu_arrays.m

If the user wants to speed up the computation by using GPUs, MATLAB provides a convenient way. Notice that the Parallel Computing Toolbox needs to be installed to enable GPU computing in MATLAB. The MATLAB function *gpuArray* is used to copy the data to GPU memory. By calling *gpuArray* supported functions, such as *fft*, the computation will automatically run on GPUs. A detailed list of *gpuArray* supported functions can be found in MATLAB documentation.

4.8 Description of Update_C.m

To model uniform corrosion and pitting corrosion, two different functions are defined in this file. In each function, the concentration is updated using the forward Euler method. Characteristic functions are also updated at every time step. The details are shown below.

Algorithm 7 Structure of update_C.m

```
Function update_C
    If the corrosion type is uniform corrosion (corrosion_type equals to 0)
        Call function uniform_corrosion_time_integration
    Else if the corrosion type is pitting corrosion (corrosion_type equals to 1)
        Call function pitting_corrosion_time_integration

Function uniform_corrosion_time_integration
    Update the characteristic functions based on the definition in Eq. (10).
    Update the concentration based on Eq. (16)
Function pitting_corrosion_time_integration
    Update the characteristic functions based on the definition in Eq. (12).
    Update the concentration based on Eq. (16)
```

4.9 Description of Dump_output.m, Dump_output_Tecplot.m, Visualization.m, and Postprocess.m

These files are used for visualizing the results and storing data while the simulation is running. In *dump_output.m*, the snapshot number, concentration, and liquid characteristic function are the inputs. These variables are stored in a structure type variable *Output*. The user could easily store more variables in *Output*. In *dump_output_Tecplot*, the node coordinates and concentration are stored. In *visualization.m*, we plot the result at a cross-section in the x–y plane. The user can modify this file to plot their desired quantities. If the users choose not to visualize during runtime, they can run the *postprocess.m* script to plot the recorded output data after the simulation is finished.

5 Corrosion Examples in 3D

In this section, we show two corrosion simulations in 3D obtained with PeriFast/Corrosion: one using uniform corrosion (without considering salt layer formation) and the other pitting corrosion (which considers salt layer formation). All figures in this section are plotted using Tecplot.

5.1 Uniform Corrosion Example in 3D

In this part, we show the results for the uniform corrosion of a 3D shape selected here to be the University of Nebraska logo, the Nebraska "N," to show that arbitrary and relatively complex shapes can be considered. The thickness of the 3D sample is 40 μm, along the

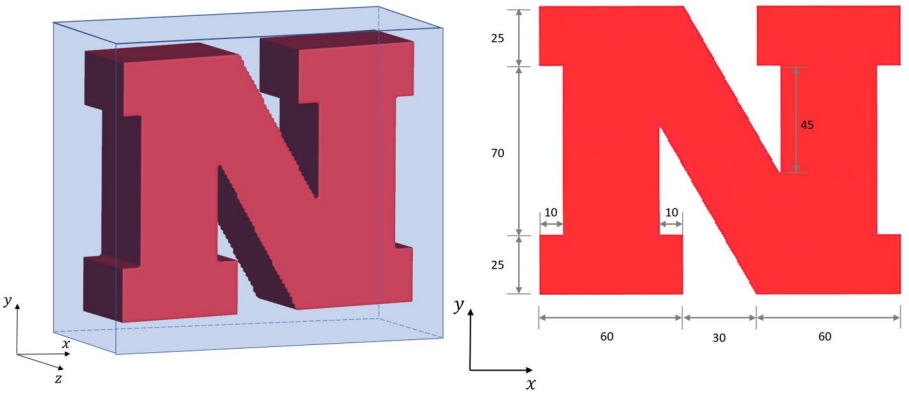

Fig. 5 Nebraska "N" sample (left) and dimensions of the cross-section in the x–y plane, units in μm (right)

z-direction. The dimensions of the cross-section in the x–y plane are shown in Fig. 5, and we define the characteristic function χ_N based on these values.

The sample is surrounded by 1 M NaCl solution. The solution is shown as the light blue box in Fig. 5, with dimensions 158 μm × 128 μm × 48 μm. Since the bulk electrolyte in many experiments normally occupies a much larger volume, zero concentration boundary conditions ($C = 0$) are applied on all surfaces of the light blue box. To make sure the horizon size is small relative to the geometrical feature in the sample, δ is set to be 4μm. In addition, this horizon size is in the range of the expected thickness of the layer affected by corrosion [29]. The blue box is extended by δ from all surfaces as the fictitious domain for enforcing volume constraints. The discretization resolution along the three Cartesian directions is $2^7, 2^7, 2^6$ which results in 1,048,576 nodes. The m-factors [30] along these directions are $\frac{\delta}{\Delta x} = 3.09$, $\frac{\delta}{\Delta y} = 3.77$, and $\frac{\delta}{\Delta z} = 4.57$.

In this example, we consider a similar uniform corrosion environment as the experiments in [6]. The metal sample is a commercial Cu plate with a purity of 99.94%. In this experiment, the 1 M NaCl solution was circulated via two pumps to avoid the local accumulation of Cu ions. Since the flowing solution flushes the dissolved Cu ions, the salt layer effect does not need to be considered here. The material properties are diffusivity $K = 1297\mu m^2 \cdot s^{-1}$, average charge number $n = 2$, and $C_{\text{Solid}} = 141,000 \text{mol}/\text{m}^3$ [31]. We choose the total time $t_{\max} = 300$s. The time step $dt = 0.8$ms is chosen based on the stability analysis in [18, 19], leading to about 300 K time steps. By taking the measured current density in [6], i is set to be 1.45 kA/m^2. The evolution of uniform corrosion is shown in Fig. 6, where the solid phase is shown at various times. A simulation movie is available in Supplementary Materials (uniform_corrosion.mp4). We can see that the corrosion occurs on all surfaces of "N" and the corners become rounded. The metal concentration map at 300 s is shown in Fig. 7, where the "trace" of the solid connection, dissolved by now, between the two vertical posts is seen in the electrolyte.

This computation is performed on a supercomputer in the Holland Computing Center of the University of Nebraska-Lincoln, with an Intel Xeon E5-2670 2.60 GHz CPU, up to 512 GB RAM per CPU, and a Tesla P100 GPU with 12 GB memory. This simulation, with no visualization during running, takes 1.4 h. More discussion on comparing the computational cost of using the FCBM or the meshfree method (with direct summation for quadrature) can be found in [18, 19, 21].

Fig. 6 Uniform corrosion of a complex shape at **a** 0 s, **b** 100 s, **c** 200 s, and **d** 300s

5.2 A pitting Corrosion Example in 3D with Multiple Growing and Merging Pits

In this example, we use the same geometry as in the last section. To model pitting corrosion, we first initiate 20 spherical pits with random locations on one of the surfaces and random radii (2 to 4 μm), as shown in Fig. 8.

The boundary conditions on the "electrolyte box" are the same as the previous uniform corrosion example. To obtain smoother pits, we need a higher resolution than that used in the uniform corrosion example. The discretization resolution along the three Cartesian directions is $2^8, 2^8, 2^6$ which results in 4,194,304 nodes. The m-factors [30] along these directions are $\frac{\delta}{\Delta x} = 6.17$, $\frac{\delta}{\Delta y} = 7.53$, and $\frac{\delta}{\Delta z} = 4.57$, respectively.

In this example, the corrosion environment is the same as the experiments in [32]. The metal sample is 304L stainless steel, and it is surrounded by 0.1 M NaCl solution. Considering salt layer formation is important in pitting corrosion (see, e.g., [33]) and is taken into account here. The material properties are diffusivity $K = 860\mu\text{m}^2 \cdot \text{s}^{-1}$, average charge number $n = 2.19$, $C_{\text{Solid}} = 142,900\text{mol/m}^3$, and NaCl solution at 25 °C $C_{\text{sat}} = 4600\text{mol/m}^3$. By taking the measured current density in [32], i is set to be 5.1 kA/m^2. We choose the total time $t_{\max} = 80\text{s}$ and time step $dt = 1\text{ms}$ which is chosen based on the stability analysis in [18, 19]. The evolution of pitting corrosion at different

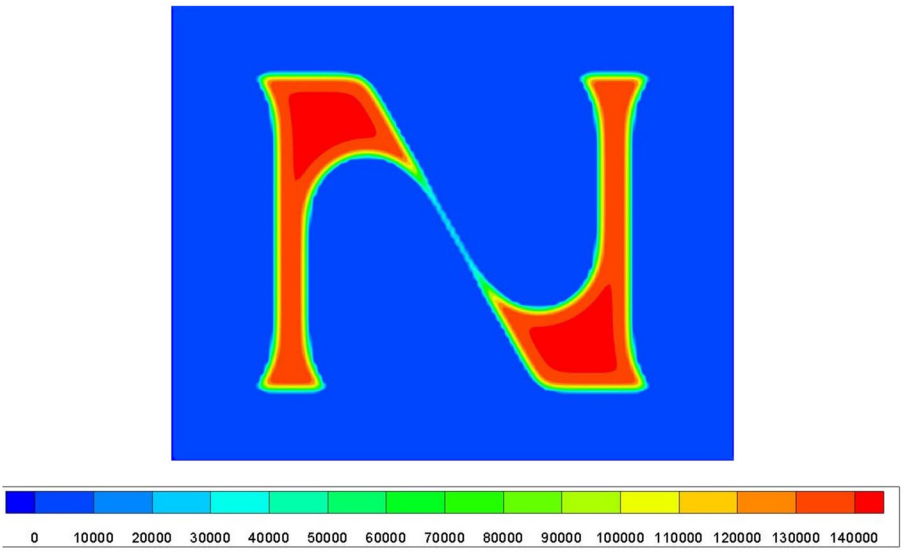

Fig. 7 Metal ion concentration in an x–y cross-section at 300s

times is shown in Fig. 9. The autonomous growth and merger of pits are observed. A simulation movie is available in Supplementary Materials (pitting_corrosion.mp4). The metal concentration in a cross-section at 80 s is shown in Fig. 10 for the section in the y–z plane through the left side of the N-letter, cutting some three merging pits.

This simulation, with no visualization processed during the run, took 2.9 h to complete on the same platform as the uniform corrosion example above.

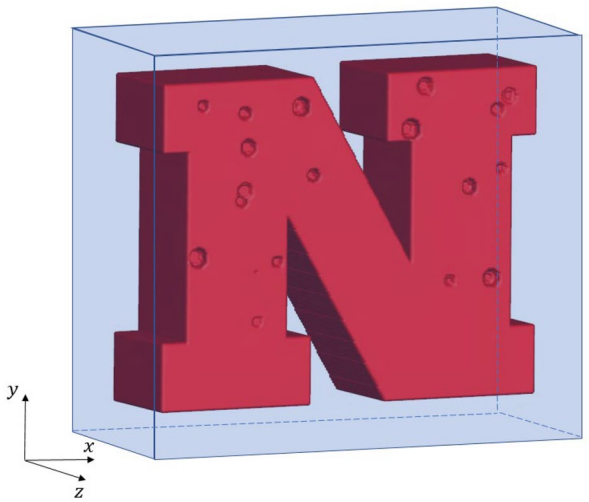

Fig. 8 Nebraska "N" sample with initial pits

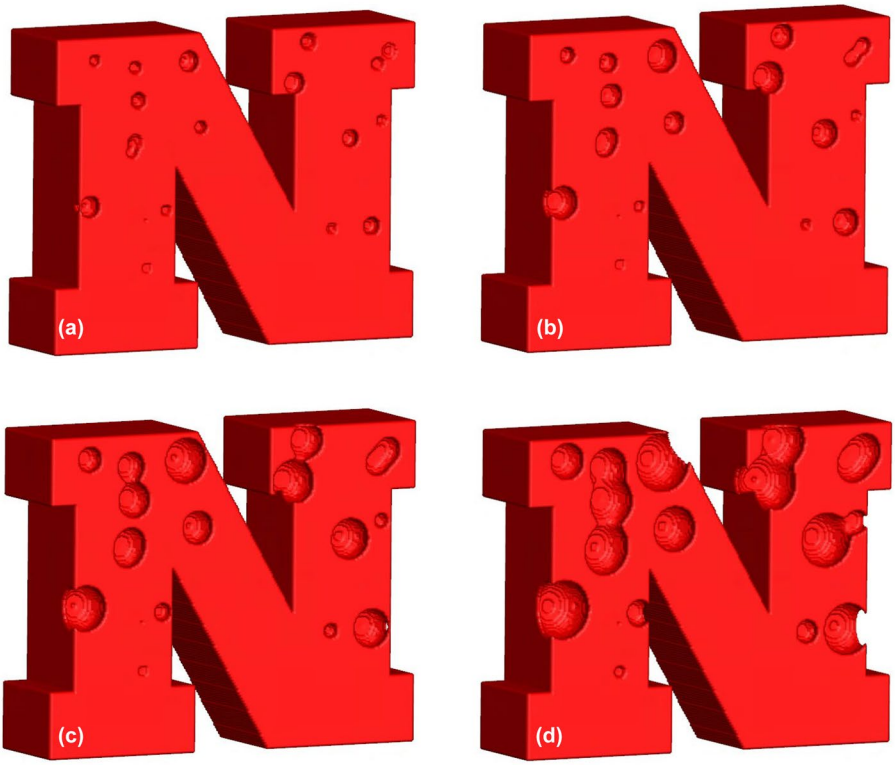

Fig. 9 Growing and merging corrosion pits on the front face of a complex shape at **a** 20 s, **b** 40 s, **c** 60 s, and **d** 80s

6 Possible Extensions of PeriFast to Other Corrosion Types

PeriFast/Corrosion introduced in this paper implements PD models for uniform and pitting corrosion. PD models for other types of corrosion have been introduced as well, for example, uniform corrosion [6], pitting corrosion with lacy covers [5], crevice corrosion [7], galvanic corrosion [8], intergranular corrosion [9], stress-dependent corrosion [6], and stress-corrosion

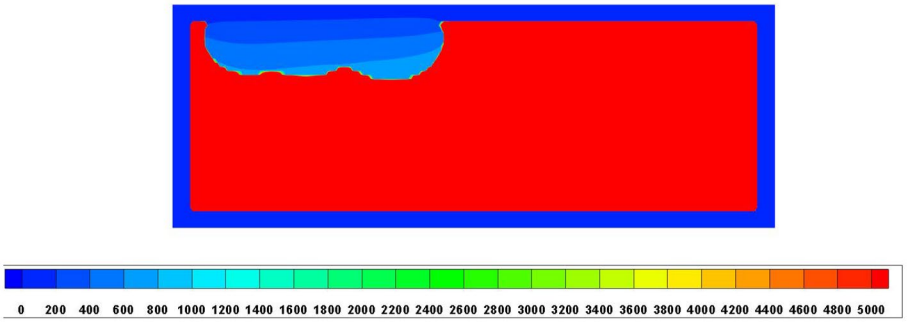

Fig. 10 Metal ion concentration in a y–z cross-section at 80s

cracking [10]. These published PD models have used the direct summation in the meshfree method with one-point Gaussian quadrature. Extending PeriFast/Corrosion to these other corrosion types requires some changes, some of which are described briefly below.

In certain materials (like stainless steel), pitting occurs with the formation of lacy covers. Part of the surface may passivate due to low pH values while corrosion continues in other regions. The corrosion process bypasses the passivated part and creates a perforated cover (the "lacy cover"). The passivation and lacy cover formation mechanism have been implemented in the pitting corrosion model and solved using FCBM [21] and could be easily added to the code.

Crevice corrosion happens when geometry restricts the flow of electrolytes in the crevice, leading to the accumulation of metal ions. Electroneutrality means that more chloride ions migrate from the bulk electrolyte into the crevice, triggering local acidification, which increases the anodic dissolution rate. To model this, the current density i depends on the concentration, and the PD formulation in Eq. (3) needs to be changed as described in [7]. Therefore, to model crevice corrosion using PeriFast/Corrosion, the user should define a new characteristic function to take into account the concentration-dependent current density.

Intergranular corrosion can significantly reduce the mechanical durability of metal alloys. Due to the local galvanic coupling of grain boundaries and grain matrix, grain boundaries are corroded preferentially, usually leading to faster dissolution along the grain boundaries. To model this, the user would need to define characteristic functions to distinguish grain boundaries from the grains themselves. The introduction of such material heterogeneities in the FCBM context could be performed as discussed in [21].

PeriFast/Corrosion can be coupled with PeriFast/Dynamics [28], which is an implementation of FCBM for dynamic elastic deformations and brittle fracture problems in 3D, to model stress-corrosion cracking and stress-dependent corrosion problems. In these types of problems, stress influences the corrosion rate and corrosion reduces the material's toughness/strength (e.g., reduce ductility), leading to early fracture. Both PeriFast/Corrosion and PeriFast/Dynamics are branches of the PeriFast code that implement the FCBM for PD models. Three different PD material models are provided in PeriFast/Dynamics: the linearized bond-based and ordinary state-based models for isotropic elastic materials and the PD correspondence model for isotropic hyperelastic materials. The current version of the code implements brittle damage models, but ductile failure can also be considered [34, 35].

7 Conclusions

In this paper, we introduced a compact and efficient MATLAB code, PeriFast/Corrosion, for simulating uniform and pitting corrosion problems. PeriFast/Corrosion uses the fast convolution-based method (FCBM) of discretization for peridynamic (PD) corrosion models.

We reviewed the PD corrosion model and the FCBM for discretization. The salt layer effect, critical in pitting corrosion, can be selected as an option in the model. Using the embedded constraint method, the FCBM applies to arbitrary domains and given boundary conditions. Compared with the direct summation used for the quadrature in the meshfree discretization of the PD model, the computational cost for FCBM is reduced from $O(NM)$ to $O(NlogN)$, and memory allocation requirements scale as $O(N)$ instead

of $O(N^2)$. N is the total number of nodes, and M is the number of nodes in the family of an arbitrary node. Due to these large gains in efficiency, one can run PD simulations of corrosion damage at larger scales and for longer time spans than when using previous discretization methods.

The structure of PeriFast/Corrosion and its modules are discussed in detail. The user of the PeriFast/Corrosion has the flexibility to modify the modules based on their needs. Two examples are included: a 3D uniform corrosion in copper and a 3D pitting corrosion in stainless steel, both running on the same nontrivial shape (the University of Nebraska "N"). The pitting corrosion example shows the autonomous growth and merger of many pits. Possible extensions of the code to a variety of other corrosion types and corrosion-related problems have been presented to show the potential of PeriFast/Corrosion.

Appendix: The Influence of the Salt Layer

In this part, using a 2D version of the FCBM pitting corrosion code [21], we investigate the influence of the salt layer by running the same pitting corrosion example with and without considering the salt layer effect. The specimen's configuration and boundary condition are shown in Fig. 11.

The size of the 2D computational sample is $1\text{mm} \times 1\text{mm}$. The material is stainless steel 304SS. The average charge number ($n = 2.19$) [36] of 304SS is calculated from the charge number of Fe, Ni, Cr, and their mole fractions. The specimen is submerged in 1 M NaCl solution. The material properties are given [37]: $C_{\text{Solid}} = 143000\text{mol}/\text{m}^3$ and$C_{\text{sat}} = 5100\text{mol}/\text{m}^3$. The initial current density in the experiment [38] is measured to be $3.8\text{A} \cdot \text{cm}^{-2}$.

The simulation results with and without the salt layer can be found in Fig. 12. We can see that the salt layer at the pit bottom influences the pit's shape and size. With the salt layer effect, the corrosion near the pit bottom is temporarily stopped, leading to the shallower pit shape. In cases when the current density is small and ions can be timely

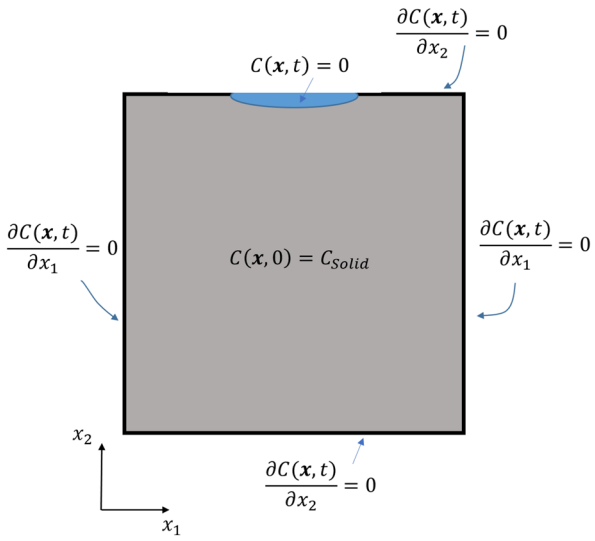

Fig. 11 Boundary conditions and initial conditions of the 2D corrosion example

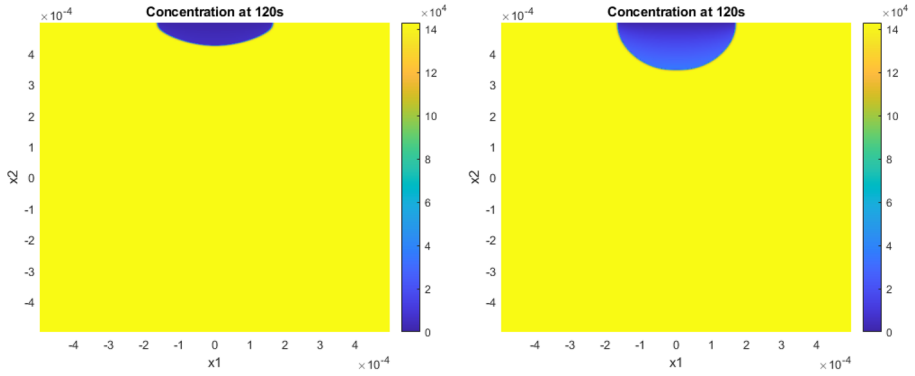

Fig. 12 PeriFast/Corrosion results with salt layer (left), and without salt layer (right). The colors represent the metal concentration

diffused out of the pit, the salt layer may not play an important role and can be ignored. A more detailed discussion of the salt layer effect can be found in [21].

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s42102-023-00098-5.

Author Contribution L. W., S. J., and F. M. implemented and tested the computer code. All authors wrote the manuscript draft. F. B. acquired funding, designed and coordinated research plan, supervised the code implementation, and edited the manuscript and computer code.

Funding This work has been supported by NSF CDS&E-CMMI grant No. 1953346 and by a Nebraska System Science award from the Nebraska Research Initiative. This work was completed utilizing the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative.

Data Availability The source code can be downloaded from https://github.com/PeriFast/Code by clicking the green "Code" button and selecting "Download ZIP". This will download all of the branches of the PeriFast code, at this time PeriFast/Corrosion and PeriFast/Dynamics, which solves dynamic fracture problems.

Declarations

Ethics Approval Not applicable.

Competing Interests The authors declare no competing interests.

References

1. PeriFast/Corrosion (2022) Retrieved from https://github.com/PeriFast/Code
2. Pistorius PC, Burstein GT (1992) Metastable pitting corrosion of stainless steel and the transition to stability. Philosophical transactions of the royal society of London. Series A: Phys Eng Sci 341(1662):531–559. https://doi.org/10.1098/rsta.1992.0114
3. Jafarzadeh S, Chen Z, Bobaru F (2019) Computational modeling of pitting corrosion. Corros Rev 37(5):419–439. https://doi.org/10.1515/corrrev-2019-0049
4. Chen Z, Bobaru F (2015) Peridynamic modeling of pitting corrosion damage. J Mech Phys Solids 78:352–381. https://doi.org/10.1016/j.jmps.2015.02.015

5. Jafarzadeh S, Chen Z, Zhao J, Bobaru F (2019) Pitting, lacy covers, and pit merger in stainless steel: 3D peridynamic models. Corros Sci 150:17–31. https://doi.org/10.1016/j.corsci.2019.01.006
6. Jafarzadeh S, Chen Z, Li S, Bobaru F (2019) A peridynamic mechano-chemical damage model for stress-assisted corrosion. Electrochimica Acta, 323:134795. https://doi.org/10.1016/j.electacta.2019.134795
7. Jafarzadeh S, Zhao J, Shakouri M, Bobaru F (2022) A peridynamic model for crevice corrosion damage. Electrochimica Acta, 401:139512. https://doi.org/10.1016/j.electacta.2021.139512
8. Zhao J, Jafarzadeh S, Rahmani M, Chen Z, Kim YR, Bobaru F (2021) A peridynamic model for galvanic corrosion and fracture. Electrochimica Acta 391:138968. https://doi.org/10.1016/j.electacta.2021.138968
9. Jafarzadeh S, Chen Z, Bobaru F (2018) Peridynamic modeling of intergranular corrosion damage. J Electrochem Soc 165(7):C362–C374. https://doi.org/10.1149/2.0821807jes
10. 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
11. 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
12. Wang L, Bobaru F (2021) Connections between the meshfree peridynamics discretization and graph Laplacian for transient diffusion problems. J Peridynamics Nonlocal Model 3(4):307–326. https://doi.org/10.1007/s42102-021-00053-2
13. Seleson P, Littlewood DJ (2016) Convergence studies in meshfree peridynamic simulations. Comput Math Appl 71(11):2432–2448. https://doi.org/10.1016/j.camwa.2015.12.021
14. Mehrmashhadi J, Wang L, Bobaru F (2019) Uncovering the dynamic fracture behavior of PMMA with peridynamics: the importance of softening at the crack tip. Eng Fracture Mech 219:106617. https://doi.org/10.1016/j.engfracmech.2019.106617
15. 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
16. 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
17. Liu W, Hong J-W (2012) A coupling approach of discretized peridynamics with finite element method. Comput Methods Appl Mech Eng 245:163–175. https://doi.org/10.1016/j.cma.2012.07.006
18. Jafarzadeh S, Larios A, Bobaru F (2020) Efficient solutions for nonlocal diffusion problems via boundary-adapted spectral methods. J Peridynamics Nonlocal Model 2:85–110. https://doi.org/10.1007/s42102-019-00026-6
19. 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
20. 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
21. Wang L, Jafarzadeh S, Larios A, Bobaru F (2023) A fast convolution-based method for peridynamics model of pitting corrosion. Submitted
22. Lopez L, Pellegrino SF (2022) A fast-convolution based space–time Chebyshev spectral method for peridynamic models. Adv Continuous Discrete Models 2022(1):70. https://doi.org/10.1186/s13662-022-03738-0
23. 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
24. Oterkus S, Madenci E, Agwai A (2014) Peridynamic thermal diffusion. J Comput Phys 265:71–96. https://doi.org/10.1016/j.jcp.2014.01.027
25. Zhao J, Jafarzadeh S, Chen Z, Bobaru F (2020) An algorithm for imposing local boundary conditions in peridynamic models on arbitrary domains. https://doi.org/10.31224/osf.io/7z8qr
26. 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
27. Isaacs HS, Cho J, Rivers ML, Sutton SR (1995) In situ X-Ray microprobe study of salt layers during anodic dissolution of stainless steel in chloride solution. J Electrochem Soc 142(4):1111. https://doi.org/10.1149/1.2044138
28. Jafarzadeh S, Mousavi F, Wang L, Bobaru F (2023) PeriFast/Dynamics: a MATLAB code for explicit fast convolution-based peridynamic analysis of deformation and fracture. J Peridynamics Nonlocal Model. https://doi.org/10.1007/s42102-023-00097-6

29. Vallabhaneni R, Stannard TJ, Kaira CS, Chawla N (2018) 3D X-ray microtomography and mechanical characterization of corrosion-induced damage in 7075 aluminium (Al) alloys. Corros Sci 139(2017):97–113. https://doi.org/10.1016/j.corsci.2018.04.046
30. Chen Z, Bobaru F (2015) Selecting the kernel in a peridynamic formulation: a study for transient heat diffusion. Comput Phys Commun 197:51–60. https://doi.org/10.1016/j.cpc.2015.08.006
31. Ribeiro ACF, Esteso MA, Lobo VMM, Valente AJM, Simoes SMN, Sobral AJFN, Burrows HD (2005) Diffusion coefficients of copper chloride in aqueous solutions at 298.15 K and 310.15 K. J Chem Eng Data 50(6):1986–1990. https://doi.org/10.1021/je050220y
32. Almuaili FA (2017) Characterisation of 3D pitting corrosion kinetics of stainless steel in chloride containing environments. Ph.D. dissertation, The University of Manchester
33. Jafarzadeh S, Chen Z, Bobaru F (2018) Peridynamic modeling of repassivation in pitting corrosion of stainless steel. Corrosion 74(4):393–414
34. Mousavi F, Jafarzadeh S, Bobaru F (2021) An ordinary state-based peridynamic elastoplastic 2D model consistent with J2 plasticity. Int J Solids Struct 229:111146. https://doi.org/10.1016/j.ijsolstr.2021.111146
35. Mousavi F, Jafarzadeh S, Bobaru F (2023) A fast convolution-based method for peridynamic models in plasticity and ductile fracture. (submitted)
36. Scheiner S, Hellmich C (2009) Finite Volume model for diffusion- and activation-controlled pitting corrosion of stainless steel. Comput Methods Appl Mech Eng 198(37–40):2898–2910. https://doi.org/10.1016/j.cma.2009.04.012
37. Scheiner S, Hellmich C (2007) Stable pitting corrosion of stainless steel as diffusion-controlled dissolution process with a sharp moving electrode boundary. Corros Sci 49(2):319–346. https://doi.org/10.1016/j.corsci.2006.03.019
38. Ernst P, Newman RC (2002) Pit growth studies in stainless steel foils. I. Introduction and pit growth kinetics. Corros Sci 44(5):927–941. https://doi.org/10.1016/S0010-938X(01)00133-0