Reformulation of Elasticity using Peridynamic Theory

Parameshwaran Pasupathy

1 Introduction

This study looks at a new approach of formulating continuum mechanics methods called the Peridynamic theory. One of the underlying assumptions of classical theory is its locality. The classical theory assumes that a material point interacts only with its immediate neighbors. As a result, the stress state at a point depends on the deformation at the point alone. The interaction of the material point is then governed by the balance laws. The mathematical framework developed based on these laws is in many ways ill-suited for scenarios that involve spontaneous formation of discontinuities such as cracks. This is due to the fact that the partial differential equations are undefined at discontinuities.

Peridynamics (PD) is a non-local extension of continuum mechanics that is compatible with the physical nature of discontinuities. It avoids the need to evaluate the partial derivatives of the deformation with respect to the spatial coordinates, instead using an integro-differential equation for the linear momentum balance. In PD, internal forces are expressed through nonlocal interactions between pairs of material points within a continuous body, and damage is part of the constitutive model.

In this study, the equations of bond-based PD theory are implemented separately in Python programming language and Fortran for a block under tension and the transverse vibration of a beam. The implementation is compared to the analytical solution and the viability of this method is evaluated with regard to the above programming schemes.

2 Methods

Essentially, PD replaces the local equilibrium equation with a non-local formulation as below [1]:

$$\nabla \cdot \sigma + \mathbf{b} = \mathbf{0} \quad \rightarrow \quad \int_{\mathscr{H}_x} \mathbf{f}(\mathbf{u}' - \mathbf{u}, \mathbf{x}' - \mathbf{x}) dV_x + \mathbf{b} = \mathbf{0}, \tag{1}$$

where σ is the stress field, $\mathbf{b}$ is the prescribed body force and $\mathbf{f}$ is the force density (per unit volume squared) that the material point $\mathbf{x}$ exerts on $\mathbf{x}$'. $\mathscr{H}_x$ is the neighborhood of $\mathbf{x}$ known as the *horizon* and $\mathbf{u}$ is the displacement field of $\mathbf{x}$ (Figure 1).

The term *bond* refers to the interaction between the material points at $\mathbf{x}$ and $\mathbf{x}$'. Bond-based PD relies on pairwise set of interactions in which the force density vectors exerted by the material points on one another are equal in magnitude and parallel to the relative position vector in the deformed state. The integral equation in (1) is always valid regardless of discontinuities. In the deformed configuration, the force density function, $\mathbf{f}$, contains all the constitutive properties of the material. For a linear elastic material, the force density function can be defined as [1, 2]

$$\mathbf{f}(\boldsymbol{u} - \boldsymbol{u}', \boldsymbol{x} - \boldsymbol{x}') = c \cdot s \frac{\boldsymbol{y}' - \boldsymbol{y}}{|\boldsymbol{y}' - \boldsymbol{y}|} \tag{2}$$

where $\boldsymbol{y} = \boldsymbol{u} + \boldsymbol{x}$ is the position of the material point in the deformed configuration. c denotes the bond constant and s is the bond stretch, defined as

$$s = \frac{|\boldsymbol{y}' - \boldsymbol{y}| - |\boldsymbol{x}' - \boldsymbol{x}|}{|\boldsymbol{x}' - \boldsymbol{x}|} \tag{3}$$

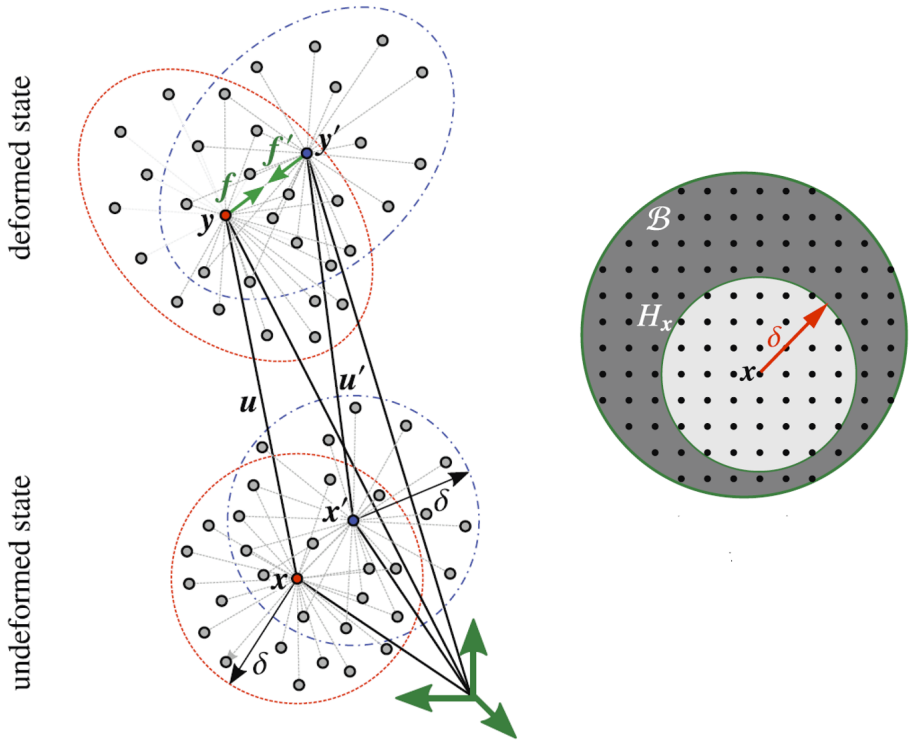

Figure 1: Illustration of the bond-based peridynamics for a material point located at x . Interactions within the neighborhood $\mathscr{H}_x$ for the material point is defined by a cut-off region known as the *horizon*. Two material points x and x' (left) and their family of material points (horizon) and their deformed state giving rise to the bond force **f**.

The bond constant c is the PD material parameter and can be expressed in terms of the material constants of CCM.

$$c = \frac{12E}{\pi\delta^4} \tag{4}$$

where E is the elastic modulus in CCM and δ is the horizon of the material point. Note that there is only one PD parameter used in the original (bond-based PD) formulation. As a result, bond-based PD automatically captures a constant Poisson's ratio of 0.33 for two-dimensional and 0.25 for three-dimensional problems. Bond-based PD therefore cannot freely define a Poisson's ratio for a material. This drawback can be overcome using a state-based PD approach.

Another disadvantage of PD is the presence of surface effects at material points near the boundary of the material. The derivation of the PD bond stiffness c is based on the assumption that the horizon of a material point $\mathscr{H}_x$ is contained completely within the bulk of a body. This assumption does not hold true for nodes within a distance δ of the body edge (Figure 2). Nodes within δ of the body edge do not possess a full non-local neighborhood. Consequently the material properties of nodes contained within the main bulk and nodes within a distance δ of the body

edge differs. This is known as the peridynamic surface effect. The surface effect in this study is corrected by modifying the pairwise force function for the surface nodes which is known as the volume correction method [3, 4].

The evolution of the displacement and pairwise force functions are computed using an explicit time integration scheme. Explicit time integration schemes are conditionally stable and become numerically unstable for large time steps Δt [3, 4]. The critical time step for the simulation is

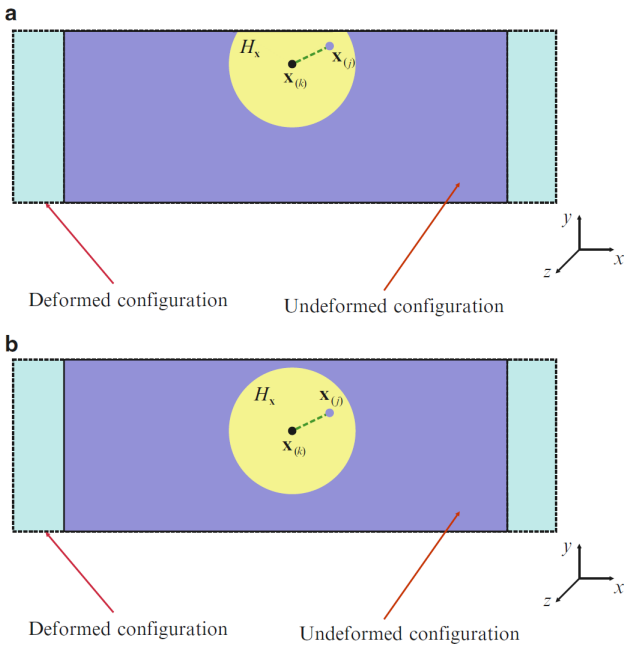

Figure 2: The horizon of two material points $x_{(k)}$ and $x_{(j)}$ with one instance contained completely within the material bulk and the other near the peridynamic body.

$$\Delta t_{crit} = \sqrt{\frac{2\rho}{\Delta V_p c_{ip}}} \tag{5}$$

A critical time step is determined for every material point in the body, and the minimum value is taken. p iterates over all the material points in the neighborhood of point i. ΔV_p is the volume of the point p, and c_{ip} is the bond stiffness between the given point i and neighbor p.

For the transverse vibration of a bar, an Euler-Cromer scheme is used to compute displacements while the quasi-static loading of block under a tensile load is computed via an adaptive dynamic relaxation (ADR) scheme coupled with the explicit solver [5]. The explicit scheme for computing displacements and updating nodal positions is shown below,

$$\begin{aligned} \ddot{\mathbf{u}}_i^n &= \frac{1}{\rho}[\sum_p \mathbf{f}(\mathbf{u}_p^n - \mathbf{u}_i^n, \mathbf{x}_p^n - \mathbf{u}_i^n) dV_p + \mathbf{b}_i^n] \\ \dot{\mathbf{u}}_i^{n+1} &= \dot{\mathbf{u}}_i^n + \ddot{\mathbf{u}}_i^n \Delta t \\ \mathbf{u}_i^{n+1} &= \mathbf{u}_i^n + \dot{\mathbf{u}}_i^n \Delta t \end{aligned} \tag{6}$$

The PD equation of motion does not contain any spatial derivatives; therefore, constraint conditions are, in general, not necessary for the solution of an integro-differential equation. However, such conditions can be imposed by prescribing constraints on displacement and velocity fields in a 'fictitious material layer' along the boundary of a nonzero volume. For ADR, the PD equation of motion is written as a set of ordinary differential equations for all material points in the system by introducing new fictitious inertia and damping terms [5 6].

3 Peridynamic Code

The numerical scheme to generate a meshless scheme (Figure 3 (left)) is developed separately in both python and Fortran with the long term goal of combining fortran's speed with python's ease of code development.

- **User input** - The material parameters, dimensions, boundary conditions and the geometry of the model are defined in a dictionary called global-variables.py. Here the user can specify if the model is being solved in 1D or 3D and the nature of the problem, namely, vibration of quasi-static tension.
- **Modules** - Depending on the input parameters the mesh-file.py script contains modules that implement the PD method.

  Calculate Material parameters such as inter-nodal spacing, the radius of the horizon.

  Create grid Computes the total number of nodes based on the mesh-discretization values and generates a grid and initializes the arrays.

  Boundary region - Depending on the user input for the application of boundary conditions, the function updates the grid with a fictitious set of material points and updates the required arrays.

  Compute material point distances and bond lengths - The function is used to compute distances between material points in order to determine the horizon and the number of material points in the horizon (family). Computation of bond lengths is necessary to determine the pairwise force function **f** between particle pairs as the simulation evolves.

  Horizon - Determines if the distance between any pair of points is within the computed horizon for the material point and establishes family members. An important issue is to adopt a method for storing the family members of the material points in order to overcome possible memory limitations. For this purpose, two different arrays can be utilized. The first array (see Array 1 in Figure 3 (right)) can store all family members of material points sequentially in a single column. The second array (see Array 2 in Figure 3 (right)) can be utilized as an indicator for the first array, so that the family members of a particular material point can be easily extracted from the first array

  Surface Correction - Implements the surface correction for the boundary material points based on the volume correction algorithm specified in [4]

The fortran code is similarly structured with subroutines for horizon esitmation, grid generation, etc, built into a module which is then called by the main program.

3.1 Testing

The folder /tests contains the tests for all the modules. The tests are written using unittest in Python. The tests are written as a class with the each method performing checks for modules

and regression tests for functions in mesh-file.py, Algorithms.py and postprocess.py. The testing strategy is to check for consistent array dimensions, output-types and regression. For instance, depending on the number of user specified grid spacing and model dimensions, the output arrays for the generated grid, boundary nodes and horizon families of each material point should be consistent. Similarly, the total nodes computed for the model should be consistent with the grid spacing in each direction and the boundary nodes specified in any particular direction.

3.2 Integration

Outside of the code structure, the package contains a noxfile to run tests and build documentation from docstrings using sphinx, pyproject.toml and a pre-commit file. Pre-commit checks for yaml conflicts, eof fixer, trailing whitespace capitalization, etc. Also pre-commit mirrors for black, isort, flake8 and pyupgrade are incorporated. Pyproject.toml includes dependencies for numpy, numba, warnings, itertools, backend-build using hatchling, dependencies for sphinx, nox and ruff.

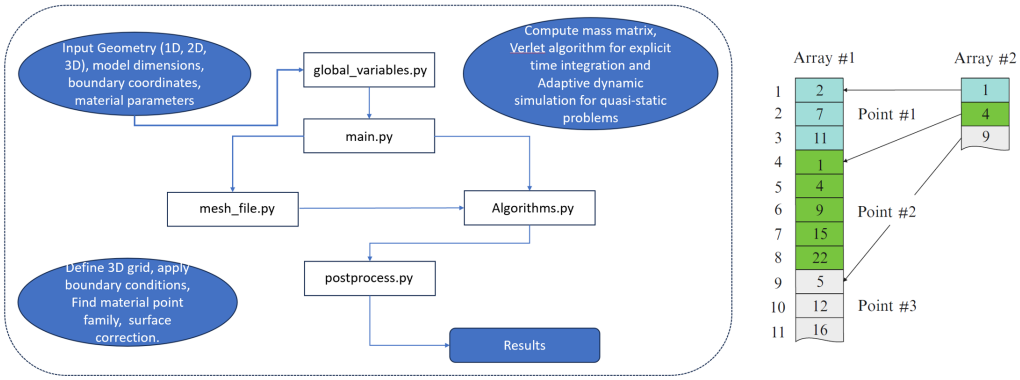

Figure 3: Schematic of the framework for implementing PD simulations (Left). Arrays used to store family information (Right). The first array stores all the family members of material point sequentially. The second array is used as an indicator for the first array. Example, second element of array 2 (4) is associated with material point 2 and is the fourth element of array 1, indicating that it is the first material point within the family of material point 2.

4 Results

The results from the PD solutions are shown in Figure 4. The results are compared with the analytical solution. It can be seen that the simulations agree very well with the analytical results. The results are plotted along the center-line of the block for elongation and Poisson contraction. For the beam vibration problem, the displacements of the material point at the center is measured with time. From the fortran simulations for the 3D block, the solution is found to be stable for a time-step size of 1000. With regard to simulation time, the python code is far slower than the fortran codes. For serial execution, on a single core with a single task on adroit cluster, the python code required 3.3 hours to run the simulation while the fortran code ran the simulations in 299 seconds. Using numba with python improved the simulation time by 30 minutes. The beam problem with 26000 timesteps required one second of execution time with fortran while the

Material	**Block**	**Bar**
E	200 GPa	200GPa
ν	0.25	0.25
BC	Left end $u_x = u_y = u_z = 0$	$u_x(x = 0) = 0$
Load	Uniform tensile load 200 MPa	0.0
Material points	[100, 10, 10]	[1000, 1, 1]
Boundary points	3 along x	3 along x
Grid spacing	0.01 m	0.001
Volume of Material Point	0.0001	1E-9
Horizon	$3.015 \cdot \Delta$	$3.015 \cdot \Delta$
Time step size	1.0 s	1.94598 x 10ˆ{-7}
Number of time steps	4000	26000
Dimensions	[1, 0.1, 0.1]	[1, 1E-3, 1E-3]
Initial displacement gradient	0.0	$u_x, x = 0.001$
Initial Velocity	0.0	$\dot{u_x} = 0$

Table 1: Material parameters for the block and beam problem

python code required 1 hour and 4 minutes. An mpi version of the fortran code is on the github repo but the code is currently encountering SIGSEV memory faults. The debugging process is ongoing and requires more time.

5 Conclusions

In this study, a PD formulation of solid mechanics is developed and implemented in python and fortran. The results are compared with the analytical solution. PD theory is able to effectively reproduce the results from CCM. With regard to the simulation time, using fortran routines is far more computationally efficient than using python. The numba version showed a marked improvement but this required a good amount of code to be rewritten as numba does not allow dynamic unpacking of dictionaries or functions with variable number of arguments. Also, further vectorization of the solver code could improve its efficiency. The mpi version of the fortran code is still being debugged for segmentation faults. The long term goal of this project is to develop this code into a full-fledged software for performing peridynamic simulations of damage in the microstructure of brain white matter. It is observed that as the brain tissue is stretched, the axonal fibers in the tissue soften undergo damage. There is evidence that the softening is due to the onset and progression of micro-cracks in the axonal fiber tracts. As described earlier, peridynamics allows for naturally incorporating discontinuities in the structure. The end goal is to combine the high efficiency of fortran and the ease of programming in python to develop a microstructural model of damage in brain white matter. The compute intensive algorithms would be written in fortran and the model build, post-processing in python. Routines such f2py could be used to interface between the two codes and make the best use of both these programming tools.

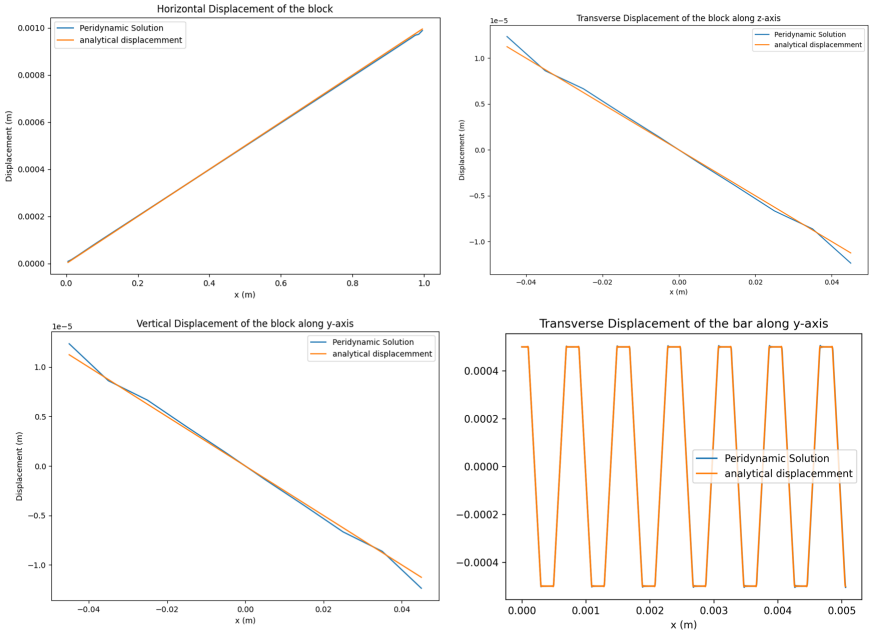

Figure 4: Results comparing the peridynamics simulations for the block in tension and beam vibration with analytical solutions. For the chosen time step and size of the horizon, there is an excellent match between the analytical and peridynamic results.

References

1. Silling SA (2000) Reformulation of elasticity theory for discontinuities and long-range forces. J Mech Phys Solids 48:175–209

2. Silling SA, Epton M, Weckner O, Xu J, Askari A (2007) Peridynamics states and constitutive modeling. J Elast 88:151–184

3. Silling SA, Askari E (2005) A meshfree method based on the peridynamic model of solid mechanics. Comput Struct 83:1526–1535

4. Seleson P, Beneddine S, Prudhomme S (2013) A force-based coupling scheme for peridynamics and classical elasticity. Comput Mater Sci. 66:34–49

5. Underwood P (1983) Dynamic relaxation. Comput Meth Trans Anal 1:245–265

6. Kilic B, Madenci E (2010) An adaptive dynamic relaxation method for quasi-static simulations using the peridynamic theory. Theor Appl Fract Mech 53:194–201