

# SANDIA REPORT

2012-7800

Unlimited Release

Printed September 2012

## Peridigm Users' Guide v1.0.0

Michael L. Parks, David J. Littlewood, John A. Mitchell, and Stewart A. Silling

Prepared by  
Sandia National Laboratories  
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



**Sandia National Laboratories**

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

**NOTICE:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from  
U.S. Department of Energy  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Telephone: (865) 576-8401  
Facsimile: (865) 576-5728  
E-Mail: [reports@adonis.osti.gov](mailto:reports@adonis.osti.gov)  
Online ordering: <http://www.osti.gov/bridge>

Available to the public from  
U.S. Department of Commerce  
National Technical Information Service  
5285 Port Royal Rd  
Springfield, VA 22161

Telephone: (800) 553-6847  
Facsimile: (703) 605-6900  
E-Mail: [orders@ntis.fedworld.gov](mailto:orders@ntis.fedworld.gov)  
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



2012-7800  
Unlimited Release  
Printed September 2012

# Peridigm Users' Guide

## v1.0.0

Michael L. Parks    David J. Littlewood  
John A. Mitchell    Stewart A. Silling

Computing Research Center  
Sandia National Laboratories  
P.O. Box 5800  
Albuquerque, NM 87185-1320

### Abstract

Peridigm is Sandia's primary open-source computational peridynamics code. It is a component software project, built largely upon Sandia's Trilinos project and Sandia's agile software components efforts. It is massively parallel, utilizes peridynamic state-based material models, Exodus/Genesis-format mesh input, Exodus-format output, and multiple material blocks. It performs explicit dynamic, implicit dynamic, and quasistatic analyses utilizing powerful nonlinear and linear solvers.

## Acknowledgments

The author acknowledges the help, support, and encouragement of John Aidun, Andy Salinger, and Randy Summers at Sandia National Laboratories.

The authors thank the following individuals for their contributions to Peridigm:

John Foster & James O'Grady (UTSA)	Elastic/plastic model with hardening, nonlinear solver development
Canio Hoffarth (ASU)	Kinetic and strain energy compute classes, friction contact model, MergeFiles script
Tracy Vogler (Sandia)	Funding and support for cylinder compression problem

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Quick Start Guide	7
1.2	Typographical Conventions	7
<b>2</b>	<b>Peridynamic Theory</b>	<b>9</b>
2.1	Basic Notation	9
2.2	State-Based Peridynamic Theory	10
2.3	Linear Peridynamic Solid (LPS) Model	11
2.4	Peridynamic Plasticity Model	12
2.5	Peridynamic Viscoelastic Model	13
2.6	Damage	15
<b>3</b>	<b>Understanding and Using Peridigm</b>	<b>17</b>
3.1	Peridigm Usage and Workflow	17
3.2	Peridigm Input Deck Overview	17
3.3	Peridigm Directory Structure	18
3.4	Peridigm Software Components	19
3.5	Downloading and Building Peridigm	20
3.6	Spatial Discretization	20
3.7	Explicit Time Integration	20
3.8	Implicit Time Integration	21
3.9	Contact	23
3.10	Damage	23
3.11	Output	23
3.12	Visualization	24
3.13	Pitfalls	24
3.14	Bugs	24
<b>4</b>	<b>Extending Peridigm</b>	<b>25</b>
4.1	Compute Classes	25
4.2	Material Models	26
4.3	Creating Tests	27
<b>5</b>	<b>A Numerical Example</b>	<b>28</b>
5.1	Problem Description and Setup	28
5.2	Writing the Peridigm Input File	28
5.3	Running Peridigm and Visualizing Output	29
	<b>References</b>	<b>33</b>

## Figures

1	Subfigure (a): Each point $\mathbf{x}$ in the body $\Omega$ interacts directly with all points in the sphere $\mathcal{H}_{\mathbf{x}}$ of radius $\delta$ (the family of $\mathbf{x}$ ). Subfigure (b): The deformation state $\mathbf{Y}$ maps a bond $\boldsymbol{\xi}$ into its deformed configuration at time $t$ . . . . .	11
2	Standard linear solid. . . . .	14
3	(a) Assuming a given bond fails at a critical stretch of $s_0$ , then the work required to break that bond is $w_0$ . (b) Summing the work required to break all bonds across a plane in (2.27) gives the energy release rate $G_0$ , an experimentally measurable quantity. . . . .	16
4	Incremental kinematics for implicit time integration. . . . .	21
5	Cylinder geometry and material properties. . . . .	28
6	Brittle cylinder before and after simulation. Color indicates damage (red = damaged, blue = undamaged). . . . .	30

## Tables

1	Notational conventions. . . . .	8
---	---------------------------------	---

# 1 Introduction

This document describes the Peridigm computational peridynamics code, Sandia’s primary open-source peridynamics code. This document complements information found at the Peridigm web site <https://software.sandia.gov/trac/peridigm>. The most up-to-date information on Peridigm and the latest version of this document can be found there.

Peridigm is a component software project, built largely upon Sandia’s Trilinos Project [4, 11]. It is massively parallel, contains peridynamic state-based material models, Exodus/Genesis-format mesh input, Exodus-format output, utilizes multiple material blocks, and provides capabilities for explicit and implicit dynamic and quasistatic analyses utilizing powerful nonlinear and linear solvers. Users interested in uncertainty quantification, optimization, calibration, sensitivity analysis, etc. for peridynamics can use Sandia’s DAKOTA project [3, 6] in conjunction with Peridigm.

In §2 we provide a brief discussion of the peridynamic theory as it motivates the implementation of Peridigm. §3 discusses the implementation of Peridigm and its capabilities and features, with §4 giving a discussion of how users can extend the capabilities of Peridigm for their own purposes. Finally, §5 presents a numerical example, walking through the setup, execution, and visualization of an example problem.

## 1.1 Quick Start Guide

For those who hate reading users’ guides, please do the following:

1. Visit the Peridigm Trac site <https://software.sandia.gov/trac/peridigm/wiki>.
2. Follow the instructions in the “Getting Started” page to download and install Peridigm and the third-party libraries required by Peridigm.
3. In the Peridigm examples directory `Peridigm/examples`, run an example input script. For example, in the `fragmenting_cylinder` directory, execute `Peridigm_fragmenting_cylinder.xml`.<sup>1</sup>
4. Follow instructions in §3.12 to visualize results.

## 1.2 Typographical Conventions

Our typographical conventions are found in Table 1. All norms  $\|\cdot\|$  are taken to be the 2-norm,  $\|\cdot\|_2$ .

---

<sup>1</sup>This example will take quite some time to complete in serial, and is recommended to run it in parallel.

**Table 1.** Notational conventions.

Notation	Example	Description
<b>Verbatim text</b>	<code>make g++</code>	Text typed at command line
<text in angle brackets>	<your platform>	User specified statement
Non-bold letter	$K, \alpha$	A scalar in $\mathbb{R}$
Bold lowercase letter	$\mathbf{x}, \boldsymbol{\xi}$	A vector in $\mathbb{R}^3$
Bold uppercase letter	$\mathbf{M}, \mathbf{J}$	A matrix in $\mathbb{R}^{n \times m}$
Underlined lowercase letter	$\underline{t}, \underline{\omega}$	Scalar state (see §2.1)
Underlined bold uppercase letter	$\underline{\mathbf{T}}, \underline{\mathbf{M}}$	Vector state (see §2.1)
Underlined blackboard bold uppercase letter	$\mathbb{D}, \mathbb{K}$	Double state (see §2.1)

## 2 Peridynamic Theory

The following is not a complete overview of peridynamics, and only introduces peridynamics concepts used in Peridigm. For more on the peridynamic theory, the reader is referred to [23, 22, 18]. To begin, we define notation.

### 2.1 Basic Notation

In the following, we utilize the notation of [22]. Referring to Figure 1(a), let the position of a given point in the reference configuration be  $\mathbf{x}$ . Let  $\mathbf{u}(\mathbf{x}, t)$  and  $\mathbf{y}(\mathbf{x}, t)$  denote the displacement and position, respectively, of the point  $\mathbf{x}$  at time  $t$ .

The vector between  $\mathbf{x}$  and any point in its family is called a bond, defined as  $\mathbf{x}' - \mathbf{x}$ . Define the relative displacement vector of two bonded points  $\mathbf{x}$  and  $\mathbf{x}'$  as  $\boldsymbol{\eta} = \mathbf{u}(\mathbf{x}', t) - \mathbf{u}(\mathbf{x}, t)$ . We note here that  $\boldsymbol{\eta}$  is time-dependent, and that  $\boldsymbol{\xi}$  is not. It follows that the relative position of the two bonded points in the current configuration can be written as  $\boldsymbol{\xi} + \boldsymbol{\eta} = \mathbf{y}(\mathbf{x}', t) - \mathbf{y}(\mathbf{x}, t)$ .

Peridynamic models are frequently written using *states*, which we briefly describe here. For a more complete discussion of states, see [22]. For our purposes, states are operators that act on vectors in  $\mathcal{H}_{\mathbf{x}}$ , defined below. A *vector state* is an operator whose image is a vector, and may be viewed as a generalization of a second-rank tensor. We write a vector state as  $\underline{\mathbf{A}} \langle \boldsymbol{\xi} \rangle$ , where the notation is understood to mean that the vector state  $\underline{\mathbf{A}}$  maps a bond vector  $\boldsymbol{\xi} \in \mathcal{H}_{\mathbf{x}}$  to a vector in  $\mathbb{R}^3$ . Similarly, a *scalar state* is an operator whose image is a scalar. We write a scalar state as  $\underline{\mathbf{a}} \langle \boldsymbol{\xi} \rangle$ , where the notation is understood to mean that the scalar state  $\underline{\mathbf{a}}$  maps a bond vector  $\boldsymbol{\xi} \in \mathcal{H}_{\mathbf{x}}$  to a scalar in  $\mathbb{R}$ . The inner product of two vector states can be written as

$$\underline{\mathbf{A}} \bullet \underline{\mathbf{B}} = \int_{\mathcal{H}_{\mathbf{x}}} \underline{\mathbf{A}} \langle \boldsymbol{\xi} \rangle \cdot \underline{\mathbf{B}} \langle \boldsymbol{\xi} \rangle dV_{\boldsymbol{\xi}}. \quad (2.1)$$

Let the symbol  $\underline{\mathbb{D}}$  denote a double state, such that  $\underline{\mathbb{D}} \langle \boldsymbol{\xi}, \boldsymbol{\zeta} \rangle$  is a second-order tensor, where  $\boldsymbol{\xi}$  and  $\boldsymbol{\zeta}$  are bonds in  $\mathcal{H}_{\mathbf{x}}$ . Let the right product of a vector state  $\underline{\mathbf{A}}$  and double state  $\underline{\mathbb{D}}$  be the vector state  $\underline{\mathbb{D}} \bullet \underline{\mathbf{A}}$  defined by

$$(\underline{\mathbb{D}} \bullet \underline{\mathbf{A}}) \langle \boldsymbol{\xi} \rangle = \int_{\mathcal{H}_{\mathbf{x}}} \underline{\mathbb{D}} \langle \boldsymbol{\xi}, \boldsymbol{\zeta} \rangle \underline{\mathbf{A}} \langle \boldsymbol{\zeta} \rangle dV_{\boldsymbol{\zeta}}. \quad (2.2)$$

We define Fréchet derivatives of functions of a vector state following the discussion in [20]. Let  $W$  be a scalar-valued function of a vector state  $\underline{\mathbf{A}}$ , such that

$$W(\underline{\mathbf{A}} + \Delta \underline{\mathbf{A}}) = W(\underline{\mathbf{A}}) + W_{\underline{\mathbf{A}}} \bullet \Delta \underline{\mathbf{A}} + o(\|\Delta \underline{\mathbf{A}}\|) \quad (2.3)$$

holds. Then,  $W$  is said to be differentiable and  $W_{\underline{\mathbf{A}}}$  is called the Fréchet derivative of  $W$ . Likewise, if  $\underline{\mathbf{S}}$  is vector state valued function of a vector state  $\underline{\mathbf{A}}$ , then

$$\underline{\mathbf{S}}(\underline{\mathbf{A}} + \Delta \underline{\mathbf{A}}) = \underline{\mathbf{S}}(\underline{\mathbf{A}}) + \underline{\mathbf{S}}_{\underline{\mathbf{A}}} \bullet \Delta \underline{\mathbf{A}} + o(\|\Delta \underline{\mathbf{A}}\|), \quad (2.4)$$

where  $\underline{\mathbf{S}}_{\mathbf{A}}$  is a double state.

Some specific states we will find useful in our discussion of peridynamics are the deformation and displacement states, which we define here. Referring to Figure 1(b), we define the deformation state  $\underline{\mathbf{Y}}[\mathbf{x}, t] \langle \mathbf{x}' - \mathbf{x} \rangle = \mathbf{y}(\mathbf{x}', t) - \mathbf{y}(\mathbf{x}, t)$ , where  $\mathbf{y}(\mathbf{x}, t)$  denotes the position of point  $\mathbf{x}$  at time  $t$ . We can define the closely related displacement state as  $\underline{\mathbf{U}}[x, t] \langle \mathbf{x}' - \mathbf{x} \rangle = \mathbf{u}(\mathbf{x}', t) - \mathbf{u}(\mathbf{x}, t)$ .

## 2.2 State-Based Peridynamic Theory

The peridynamic equation of motion is

$$\rho(\mathbf{x})\ddot{\mathbf{u}}(\mathbf{x}, t) = \int_{\mathcal{H}_{\mathbf{x}}} \{ \underline{\mathbf{T}}[\mathbf{x}, t] \langle \mathbf{x}' - \mathbf{x} \rangle - \underline{\mathbf{T}}[\mathbf{x}', t] \langle \mathbf{x} - \mathbf{x}' \rangle \} dV_{\mathbf{x}'} + \mathbf{b}(\mathbf{x}, t), \quad (2.5)$$

where  $\rho$  represents the mass density,  $\underline{\mathbf{T}}$  the force state described below, and  $\mathbf{b}$  an external body force density. The corresponding peristatic equation is

$$- \int_{\mathcal{H}_{\mathbf{x}}} \{ \underline{\mathbf{T}}[\mathbf{x}, t] \langle \mathbf{x}' - \mathbf{x} \rangle - \underline{\mathbf{T}}[\mathbf{x}', t] \langle \mathbf{x} - \mathbf{x}' \rangle \} dV_{\mathbf{x}'} = \mathbf{b}(\mathbf{x}, t). \quad (2.6)$$

Referring to Figure 1(a), (2.5) supposes that point  $\mathbf{x}$  interacts directly and nonlocally with all points that lie within a distance  $\delta$  of  $\mathbf{x}$ . We call  $\delta$  the *horizon*, and denote the spherical region of radius  $\delta$  centered at  $\mathbf{x}$  as  $\mathcal{H}_{\mathbf{x}}$ , the *family* of  $\mathbf{x}$ . Conditions on  $\underline{\mathbf{T}}$  for which (2.5) satisfies the balance of linear and angular momentum are given in [22]. From (2.5), we see that the deformation at  $\mathbf{x}$  depends collectively upon the deformations of  $\mathcal{H}_{\mathbf{x}}$  and  $\mathcal{H}_{\mathbf{x}'}$ .

$\underline{\mathbf{T}}[\mathbf{x}, t]$  is the force state. It is a vector state that is a function of both space  $\mathbf{x}$  and time  $t$  that maps bonds onto bond force densities.  $\underline{\mathbf{T}}$  is determined by the constitutive model  $\underline{\mathbf{T}} = \widehat{\mathbf{T}}(\underline{\mathbf{Y}})$ , where  $\widehat{\mathbf{T}}$  maps the deformation state to the force state. A material is elastic if there exists a differentiable scalar valued  $W$  such that

$$\underline{\mathbf{T}} = \widehat{\mathbf{T}}(\underline{\mathbf{Y}}) = W_{\underline{\mathbf{Y}}}. \quad (2.7)$$

Here,  $W$  is called the *strain energy density function* for the material. Many peridynamic analogues of classical material models have been developed, including linear elastic [22], elastic-plastic [14], and viscoelastic [13] models. Additionally, it is possible to wrap and utilize classical material models, using them to define a peridynamic force state  $\underline{\mathbf{T}}$ .

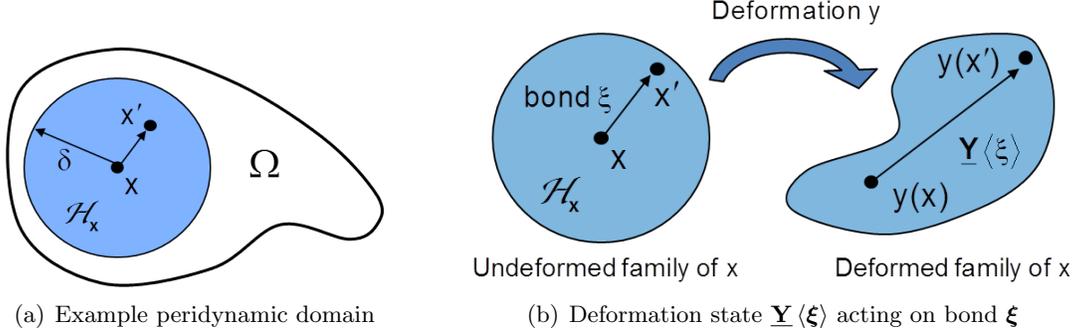
We consider only force vector states that can be written as

$$\underline{\mathbf{T}} = \underline{t} \underline{\mathbf{M}},$$

with  $\underline{t}$  a *scalar force state* and  $\underline{\mathbf{M}}$  the *deformed direction vector state*, defined by

$$\underline{\mathbf{M}}(\underline{\xi}) = \begin{cases} \frac{\underline{\xi} + \underline{\eta}}{\|\underline{\xi} + \underline{\eta}\|} & \|\underline{\xi} + \underline{\eta}\| \neq 0 \\ \mathbf{0} & \text{otherwise} \end{cases}. \quad (2.8)$$

Such force states correspond to so-called *ordinary* materials ([22]). These are the materials for which the force between any two interacting points  $\mathbf{x}$  and  $\mathbf{x}'$  acts along the line between the points.



**Figure 1.** Subfigure (a): Each point  $\mathbf{x}$  in the body  $\Omega$  interacts directly with all points in the sphere  $\mathcal{H}_{\mathbf{x}}$  of radius  $\delta$  (the family of  $\mathbf{x}$ ). Subfigure (b): The deformation state  $\underline{\mathbf{Y}}$  maps a bond  $\xi$  into its deformed configuration at time  $t$ .

All peridynamic material models utilize a nonnegative scalar state  $\underline{\omega}$ , known as an *influence function* [22, Defn. 3.2]. For more on influence functions, see [17]. If an influence function  $\underline{\omega}$  depends only upon the scalar  $\|\xi\|$ , (i.e.,  $\underline{\omega}(\xi) = \underline{\omega}(\|\xi\|)$ ), then  $\underline{\omega}$  is a spherical influence function. In the Peridigm implementation of many material models, the influence function  $\underline{\omega}(\|\xi\|) = 1$  is used. However, the user can define their own influence function by altering the definition of `OMEGA` in the appropriate method in the `src/materials` directory. Both spherical and non-spherical influence functions (e.g., isotropic and non-isotropic) influence functions are permitted.

### 2.3 Linear Peridynamic Solid (LPS) Model

We summarize the linear peridynamic solid (LPS) material model. For complete details on this model, the reader is referred to [22]. This model is a nonlocal analogue to a classical linear elastic isotropic material. The elastic properties of a classical linear elastic isotropic material are determined by (for example) the bulk and shear moduli. For the LPS model, the elastic properties are analogously determined by the bulk and shear moduli, along with the horizon  $\delta$ .

The LPS model has a force scalar state

$$\underline{t} = \frac{3K\theta}{m} \underline{\omega} \underline{x} + \alpha \underline{\omega} \underline{e}^d, \quad (2.9)$$

with  $K$  the bulk modulus and  $\alpha$  related to the shear modulus  $G$  as

$$\alpha = \frac{15G}{m}.$$

The remaining components of the model are described as follows. Define the reference position scalar state  $\underline{x}$  so that  $\underline{x}(\xi) = \|\xi\|$ . Then, the weighted volume  $m$  is defined as

$$m[\mathbf{x}] = \int_{\mathcal{H}_{\mathbf{x}}} \underline{\omega}(\xi) \underline{x}(\xi) \underline{x}(\xi) dV_{\xi}. \quad (2.10)$$

Let

$$\underline{e}[\mathbf{x}, t] \langle \boldsymbol{\xi} \rangle = \|\boldsymbol{\xi} + \boldsymbol{\eta}\| - \|\boldsymbol{\xi}\|$$

be the extension scalar state, and

$$\theta[\mathbf{x}, t] = \frac{3}{m[\mathbf{x}]} \int_{\mathcal{H}_{\mathbf{x}}} \underline{\omega} \langle \boldsymbol{\xi} \rangle \underline{x} \langle \boldsymbol{\xi} \rangle \underline{e}[\mathbf{x}, t] \langle \boldsymbol{\xi} \rangle dV_{\boldsymbol{\xi}}$$

be the dilatation. The isotropic and deviatoric parts of the extension scalar state are defined, respectively, as

$$\underline{e}^i = \frac{\theta \underline{x}}{3}, \quad \underline{e}^d = \underline{e} - \underline{e}^i,$$

where the arguments of the state functions and the vectors on which they operate are omitted for simplicity. We note that the LPS model is linear in the dilatation  $\theta$ , and in the deviatoric part of the extension  $\underline{e}^d$ . The influence function is  $\underline{\omega}$ , as described in §2.2. For a spherical influence function, the LPS model is isotropic [22, Prop. 14.1].

*Remark 2.1.* The weighted volume  $m$  is time-independent, and does not change as bonds break. It is computed with respect to the bond family defined at the reference (initial) configuration.

## 2.4 Peridynamic Plasticity Model

We summarize the ordinary, state-based plasticity model as presented in [14]. For complete details on this model, the reader is referred to [14, 22]. This model is a logical extension of the elastic LPS model of §2.3 to an elastic perfectly-plastic material. The material properties for the plasticity model are the bulk and shear moduli, the yield stress, and the horizon  $\delta$ .

The peridynamic plasticity model additively decomposes the deviatoric extension state  $\underline{e}^d$  of (2.9) into elastic and plastic parts, as

$$\underline{e}^d = \underline{e}^{de} + \underline{e}^{dp}. \quad (2.11)$$

Motivated by the observation that for many ductile materials plastic deformations are independent of the pressure, the isotropic elastic constitutive model given in (2.9) is written using the additive decomposition as:

$$\underline{t} = \frac{3K\theta}{m} \underline{\omega} \underline{x} + \alpha \underline{\omega} (\underline{e}^d - \underline{e}^{dp}) \quad (2.12)$$

$$= \underline{t}^i + \underline{t}^d, \quad (2.13)$$

where  $\underline{t}^i$  and  $\underline{t}^d$  are the scalar volumetric and deviatoric force states, respectively. A corresponding rate form of the above equation can also be written. In order to utilize the elastic constitutive relation for plasticity calculations, a scalar valued function  $f$ , called the yield function, is used to define a set of allowable scalar deviatoric force states as

$$f(\underline{t}^d) = \psi(\underline{t}^d) - \psi_0 < 0, \quad (2.14)$$

where  $\psi(\underline{t}^d) = \frac{1}{2}\|\underline{t}^d\|^2$ . Note that  $f$  does not include hardening, analogous to local perfect-plasticity, where  $\psi_0$  is a positive constant representing the yield (flow) point of the material. The plastic flow rule is defined

$$\underline{\dot{e}}^{\text{dp}} = \lambda \nabla^d \psi, \quad (2.15)$$

where  $\nabla^d \psi$  is the constrained Fréchet derivative of  $\psi$  and  $\lambda \geq 0$  is the so-called consistency parameter. Note that  $\nabla^d \psi$  produces functions with no dilatation.

When force states  $\underline{t}^d$  are such that  $f(\underline{t}^d) < 0$ , there is no change in the plastic deformation,  $\underline{\dot{e}}^{\text{dp}} = 0$ , and the material is instantaneously elastic. Likewise, force states  $\underline{t}^d$  are inadmissible if  $f(\underline{t}^d) > 0$ , and so plastic deformations only occur when  $f(\underline{t}^d) = 0$ . The value of the consistency parameter  $\lambda$  is non-negative and correlated with the rate of plastic deformation. When deformations are elastic,  $\lambda = 0$ , but otherwise  $\lambda > 0$  under plastic deformations. The consistency parameter and the yield function taken together are required to satisfy the Kuhn-Tucker complementary conditions

$$\lambda \geq 0, \quad f(\underline{t}^d) \leq 0, \quad \lambda f(\underline{t}^d) = 0. \quad (2.16)$$

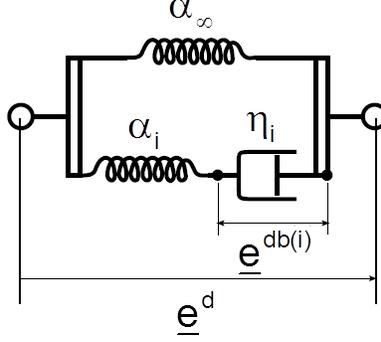
The above conditions imply that when  $f(\underline{t}^d) < 0$ , the consistency parameter  $\lambda$  must be zero. Under this condition, deformations are elastic. When  $\lambda > 0$ , admissibility of scalar force states implies that  $f(\underline{t}^d) = 0$ . Under these conditions, the force state is said to be on the yield surface, and must also persist on the yield surface. Therefore  $\dot{f}(\underline{t}^d) = 0$ . This is a plastic loading scenario. An unloading condition occurs if  $f(\underline{t}^d) = 0$  and  $\dot{f}(\underline{t}^d) < 0$ . In this case, we also have that  $\lambda = 0$ . Combining these conditions (loading and unloading) gives the consistency condition,

$$\lambda \dot{f}(\underline{t}^d) = 0. \quad (2.17)$$

A backward Euler scheme is used to incrementally integrate the deviatoric force state  $\underline{e}^{\text{dp}}$  in (2.12). See [14] for the details, as well as a derivation of  $\psi_0$  and a proof that (2.12) is thermodynamically consistent.

## 2.5 Peridynamic Viscoelastic Model

We summarize the ordinary, state-based viscoelastic model as presented in [13]. For complete details on this model, the reader is referred to [13]. A viscoelastic model is a logical intermediate between a fluid and a solid. In a viscous fluid, there is little or no elastic resistance to shear, but resistance to compressive volumetric deformations. In an elastic solid, there is elastic resistance to both shear and volumetric deformations. The peridynamic viscoelastic solid is conceptually similar to the plasticity model in §2.4, but adds viscous terms to deviatoric portion of extension state, but the bulk response remains elastic. The model uses internal state variables which are analogous to back strain(s) in the local theory. In its most elementary form, this model is conceptually similar to the standard linear solid used in the local theory. The material properties of this viscoelastic model are the bulk and shear moduli, the horizon  $\delta$ , and the parameters  $\eta_i$  and  $\tau_i^b$ , defined below.



**Figure 2.** Standard linear solid.

The total deviatoric extension state  $\underline{d}^d$  in (2.9) is additively decomposed into elastic and back parts as

$$\underline{e}^d = \underline{e}^{de} + \underline{e}^{db(i)}. \quad (2.18)$$

Under assumptions given in [13], we may express the peridynamic viscoelastic scalar force state as

$$\underline{t} = \frac{3K\theta}{m} \underline{\omega} \underline{x} + (\alpha_\infty + \alpha_i) \underline{\omega} \underline{e}^d - \alpha_i \underline{\omega} \underline{e}^{db(i)} \quad (2.19)$$

$$= \underline{t}^i + \underline{t}^d, \quad (2.20)$$

where  $\underline{t}^i$  and  $\underline{t}^d$  are the scalar volumetric and deviatoric force states, respectively. The elastic parameters  $\alpha_\infty$  and  $\alpha_i$  are defined as in a standard linear solid (see Figure 2)

$$\underline{\alpha} = \frac{15\mu}{m} \quad (2.21)$$

$$= \alpha_\infty + \alpha_i, \quad (2.22)$$

where  $\mu$  is the elastic shear modulus and  $m$  is the weighted volume. To prevent unbounded creep of this model, we require  $\alpha_\infty > 0$ . It is assumed that  $\alpha_i > 0$ , otherwise the model degenerates to a Kelvin model, which is not practical for implementation in explicit or implicit codes with kinematically driven constitutive models. Therefore, the additional constraint

$$0 < \alpha_i < \alpha \quad (2.23)$$

is required.

The evolution equation for the scalar deviatoric back extension is given based upon the following assumptions:

$$\underline{t}^d = \eta_i \dot{\underline{e}}^{db(i)} \quad (2.24)$$

$$= \alpha_i (\underline{e}^d - \underline{e}^{db(i)}), \quad (2.25)$$

where  $\underline{t}^{\text{db}(i)}$  is the scalar deviatoric force state in the dashpot and, by Newton's 3rd law, also the force in the adjoining spring. Therefore, the evolution equation is

$$\dot{\underline{e}}^{\text{db}(i)} = \frac{1}{\tau_i^b} (\underline{e}^{\text{d}} - \underline{e}^{\text{db}(i)}), \quad (2.26)$$

where  $\tau_i^b = \eta_i / \alpha_i$  is a time constant associated with the material response. The state of viscoelastic materials is history dependent. It is assumed that  $\underline{e}^{\text{db}(i)} \rightarrow 0$  in the limit as  $t \rightarrow -\infty$ . Selection of  $\tau_i^b$  is constrained by thermodynamics, and expected or relevant material response. Note that we restrict  $\eta_i > 0$ , as  $\eta < 0$  would cause the viscous effect not to oppose motion, and  $\eta = 0$ , degenerates to two springs in parallel, causing  $1/\tau_i^b$  to become undefined. Thus,  $\tau_i^b$  is positive since  $\eta_i$  is positive. Please see [14] for the details on the time integration scheme used to integrate the evolution equation (2.26) as well as proof that (2.19) is thermodynamically consistent.

## 2.6 Damage

Two material points in a body within each other's horizon in the reference configuration may stop interacting if damage is incurred. This can be represented by removing these points from each other's family. When this occurs, the removed bond carries no force, and so the remaining bonds must bear that load. This increased load makes it more likely that these other bonds will break, leading to progressive failure. Cracks propagate through this process of bond breakage and load redistribution.

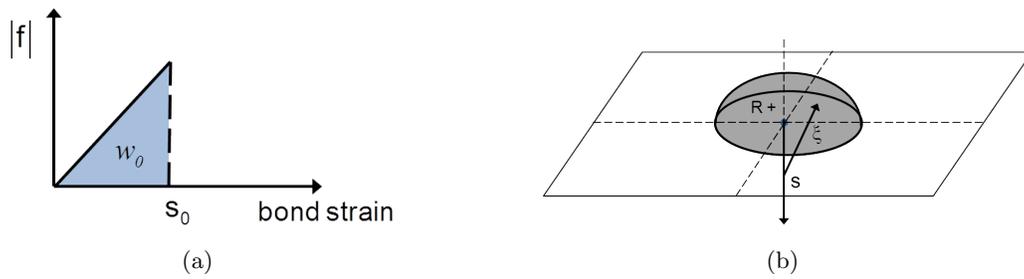
Several different criteria for bond breaking have been proposed. For example, see [9] for a discussion of a bond failure criterion based upon the rate dependent crack initiation toughness of a material. However, we discuss here only one criterion for bond breaking, called the *critical stretch* criterion [21]. Referring to Figure 3(a), suppose that the work required to break bond  $\boldsymbol{\xi}$  is  $w_0(\boldsymbol{\xi})$ . Figure 3(b) shows that the energy release rate is found by summing this work per unit crack area, giving

$$G_0 = \int_0^\delta \int_{R^+} w_0(\boldsymbol{\xi}) dV_{\boldsymbol{\xi}} ds. \quad (2.27)$$

This allows us to solve for the critical stretch  $s_0$  in terms of the energy release rate  $G_0$ , where  $G_0$  is an experimentally determinable quantity.

Following [21], we define for every bond  $\boldsymbol{\xi}$  a boolean variable  $\mu$  that is zero if a bond has broken, and one otherwise. We define the damage at a point  $\mathbf{x}$  as

$$\varphi(\mathbf{x}, t) = 1 - \frac{\int_{\mathcal{H}_{\mathbf{x}}} \mu(\boldsymbol{\xi}, t) dV_{\boldsymbol{\xi}}}{\int_{\mathcal{H}_{\mathbf{x}}} dV_{\boldsymbol{\xi}}}. \quad (2.28)$$



**Figure 3.** (a) Assuming a given bond fails at a critical stretch of  $s_0$ , then the work required to break that bond is  $w_0$ . (b) Summing the work required to break all bonds across a plane in (2.27) gives the energy release rate  $G_0$ , an experimentally measurable quantity.

## 3 Understanding and Using Peridigm

In this section we overview the capabilities and features of Peridigm. The most up-to-date information on Peridigm can be found at <https://software.sandia.gov/trac/peridigm>.

### 3.1 Peridigm Usage and Workflow

After building Peridigm (see §3.5), execute Peridigm with

```
Peridigm Input.xml
```

where `Input.xml` is a properly formatted Peridigm input deck (see §3.2).

Typical usage of Peridigm follows this general outline:

1. (Optional) Generate input mesh with mesh-generation tool, such as CUBIT [2].
2. Write Peridigm input deck. (Use Peridigm internal mesh generator if not using externally generated input mesh.)
3. Run Peridigm.
4. Merge Exodus output files (for parallel runs) with the python script `scripts/MergeFiles.py`.
5. Visualize and analyze results using visualization tool, such as Paraview [12].

Detail on each of these steps is provided in the sections below, with a complete example contained in §5.

### 3.2 Peridigm Input Deck Overview

A Peridigm input deck is an XML-formatted text file utilizing Teuchos ParameterLists [5, p.86]. We provide a brief overview of the structure of a Peridigm input deck below. A complete and up-to-date description with examples can be found at the Peridigm trac site <https://software.sandia.gov/trac/peridigm>. A working example input deck is presented in §5. See the tests and examples distributed with Peridigm for examples of other use cases.

A Peridigm input deck takes the form

```
<ParameterList>
  [Discretization Section]
  [Materials Section]
  [Damage Models Section]
  [Blocks Section]
  [Contact Section]
  [Boundary Conditions Section]
  [Solver Section]
```

[Output Section]  
</ParameterList>

We elaborate on each of these sections below.

**Discretization Section** Contains filename of input mesh, or arguments to Peridigm internal mesh generator.

**Materials Section** Contains the names of the material models used and arguments for their constitutive parameters.

**Damage Models Section** Contains the names of the damage models used and arguments for their constitutive parameters.

**Blocks Section** Contains a listing of the material blocks, associating each block with material and damage models.

**Contact Section** Contains a listing of the contact models and the kind of contact model to be applied when two blocks or materials come into contact in a simulation.

**Boundary Conditions Section** Contains the initial and boundary conditions for the simulation.

**Solver Section** Contains the solver to be used (explicit dynamics, implicit dynamics, or quasistatics) along with solver parameters.

**Output Section** Contains the output filename and output frequency along with a listing of the variables to be output.

### 3.3 Peridigm Directory Structure

The directory structure of Peridigm takes the following form:

**doc** Contains configuration file for Doxygen – Used to build Peridigm Doxygen documentation directly from source code.

**examples** Contains input decks and mesh files for several demonstration computations.

**scripts** Contains helpful python script used by the build system as well as the output

**src** Contains all source code and unit tests.

**test** Contains verification and regression tests.

Within the **src** directory, the following directories exist:

**compute** Contains source code and unit tests for the compute classes. See §4.1 to write your own.

**contact** Contains source code and unit tests for the contact models.

**core** Contains source code and units for core Peridigm functionality.

**damage** Contains source code and units for damage models.

**evaluators** Contains source code for the Phalanx evaluators [15].

**io** Contains source code for mesh input, discretization tools, and output managers.

**materials** Contains source code and unit tests for the Peridigm materials library. See §4.2 to write your own.

**muParser** Contains source code for muParser, a freeware function parser library written by Ingo Berg [8].

All Peridigm source code contains markup for Doxygen. Doxygen documentation for the entire Peridigm project may be found from the Peridigm trac site, <https://software.sandia.gov/trac/peridigm>.

The compute class and material model interfaces have been written to be easily user extensible, allowing users to easily compute any quantity of interest to them, or to easily deploy their own new material model within Peridigm. See §4 for more.

### 3.4 Peridigm Software Components

Peridigm is a component software project, built largely upon Sandia’s Trilinos project and Sandia’s agile software components efforts. We briefly describe the Trilinos Project here, showing some of the Trilinos packages consumed by Peridigm.

The Trilinos Project is an effort by Sandia National Laboratories to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics research and engineering problems [11]. Please see <http://trilinos.sandia.gov> for more. Trilinos is a collection of loosely coupled interoperable packages. Each Trilinos package focuses on a specific capability, such as linear solvers, preconditioners, nonlinear solvers, parallel linear algebra, etc. Application developers can utilize capabilities from various Trilinos packages to create an end-user application, focusing on the specific details of their application rather than reimplementing foundational capabilities.

Peridigm uses many packages from Trilinos, including:

**Belos** next-generation iterative linear solvers

**Epetra** serial and parallel linear algebra libraries

**EpetraExt** extensions to Epetra, such as matrix and vector i/o

**IFPACK** a suite of object-oriented algebraic preconditioners for preconditioned iterative solvers

**NOX** nonlinear solvers

**Phalanx** local field evaluation kernel

**Shards** common tool suite for for numerical and topological data

**STK** the Sierra ToolKit, providing an unstructured in-memory, parallel-distributed mesh database

**SEACAS** the The Sandia Engineering Analysis Code Access System, proving tools for the reading, writing, and manipulating Exodus database files

**Teuchos** common tools suite, including BLAS/LAPACK wrappers, smart pointers, parameter lists, and XML parsers

**Zoltan** parallel dynamic load balancing and related services

Peridigm inherits many standards and practices from the Trilinos Project. This includes use of CMake, a family of tools that can assist with builds across multiple platforms and regression testing across a set of target platforms [1]. Peridigm developers are also required to run a pre-checkin test suite before pushing new code to the repository. Peridigm also utilizes continuous integration testing. A project webpage is maintained at <https://software.sandia.gov/trac/peridigm>, and a user mailing list at [peridigm-users@software.sandia.gov](mailto:peridigm-users@software.sandia.gov).

### 3.5 Downloading and Building Peridigm

Please see the “Getting Started” link on the Peridigm trac site <https://software.sandia.gov/trac/peridigm> for the latest instructions on how to download and build Peridigm and the software components that it depends upon.

After building and installing Peridigm, please run `ctest` in your build directory to run all tests and confirm that you have a clean build.

### 3.6 Spatial Discretization

The region defining a peridynamic material is discretized into elements with some characteristic dimension  $\Delta x_i$ , where each element  $i$  possesses some volume  $V_i$ . All fields are assumed to be constant over an element, analogous to a midpoint quadrature discretization of the strong form equation (2.5). For any element  $i$ , let  $\mathcal{F}_i$  denote the family of elements for which element  $i$  shares a bond in the reference configuration. That is,

$$\mathcal{F}_i = \{p \mid \|\mathbf{x}_p - \mathbf{x}_i\| \leq \delta\}. \quad (3.1)$$

The discretized equation of motion for element  $i$  replaces (2.5) with the semi-discrete equation

$$\rho_i \ddot{\mathbf{u}}_i = \sum_{p \in \mathcal{F}_i} \{ \underline{\mathbf{T}}[\mathbf{x}_i, t] \langle \mathbf{x}'_p - \mathbf{x}_i \rangle - \underline{\mathbf{T}}[\mathbf{x}_p, t] \langle \mathbf{x}_i - \mathbf{x}_p \rangle \} V_p + \mathbf{b}_i^n, \quad (3.2)$$

where subscripts denote the element number. We may express the semi-discrete equation of motion for the entire body as

$$\mathbf{M} \ddot{\mathbf{u}} = \mathbf{f}(\mathbf{u}) + \mathbf{b}(\mathbf{u}), \quad (3.3)$$

where

$$\mathbf{M} = \begin{bmatrix} \rho_1 & & & \\ & \rho_2 & & \\ & & \ddots & \\ & & & \ddots \end{bmatrix} \quad (3.4)$$

is the peridynamic “mass matrix” under this discretization – a diagonal matrix of the densities of each element.

### 3.7 Explicit Time Integration

In the following sections, we denote the time at timestep  $n$  as  $t^n$ . The displacement vector  $\mathbf{u}$  at time  $t^n$  is written as  $\mathbf{u}^n = \mathbf{u}(\mathbf{x}, t^n)$ , and similarly for the other field variables. At timestep  $n$ , (3.3) is written as

$$\mathbf{M} \ddot{\mathbf{u}}^n = \mathbf{f}^n(\mathbf{u}^n) + \mathbf{b}^n(\mathbf{u}^n). \quad (3.5)$$

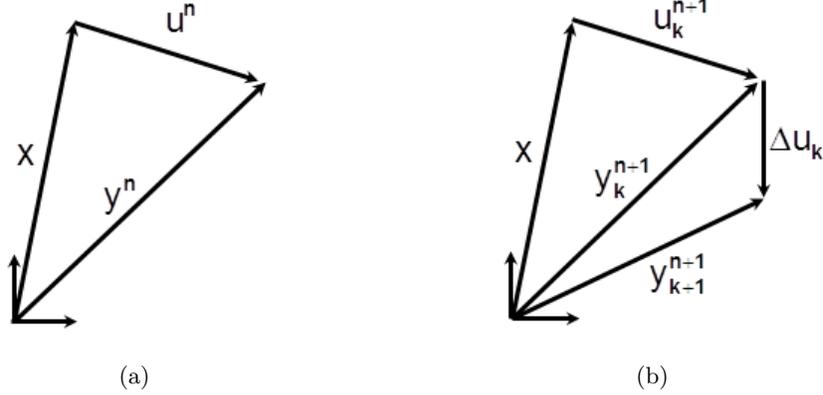
A velocity-Verlet scheme is used for explicit time integration in Peridigm. Both the position and velocity of an element are stored explicitly. The velocity-Verlet scheme is generally expressed in three steps, as shown in Algorithm 1. The internal and external force densities are reevaluated at the new displacements  $\mathbf{u}^{n+1}$  between steps 2 and 3.

---

**Algorithm 1** Velocity Verlet
 

---

- 1:  $\mathbf{v}^{n+1/2} = \mathbf{v}^n + \frac{\Delta t}{2} \mathbf{M}^{-1}(\mathbf{f}^n + \mathbf{b}^n)$
  - 2:  $\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \mathbf{v}^{n+1/2}$
  - 3:  $\mathbf{v}^{n+1} = \mathbf{v}^{n+1/2} + \frac{\Delta t}{2} \mathbf{M}^{-1}(\mathbf{f}^{n+1} + \mathbf{b}^{n+1})$
- 



**Figure 4.** Incremental kinematics for implicit time integration.

### 3.8 Implicit Time Integration

Peridigm employs the familiar Newmark- $\beta$  method [7, §6.3.3] which we repeat here for completeness, following the discussion in [14]. The displacement and velocity at timestep  $n + 1$  are computed as

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \dot{\mathbf{u}}^n \Delta t + \left[ \left( \frac{1}{2} - \beta \right) \ddot{\mathbf{u}}^n + \beta \ddot{\mathbf{u}}^{n+1} \right] (\Delta t)^2, \quad (3.6)$$

$$\dot{\mathbf{u}}^{n+1} = \dot{\mathbf{u}}^n + [(1 - \alpha) \ddot{\mathbf{u}}^n + \alpha \ddot{\mathbf{u}}^{n+1}] \Delta t, \quad (3.7)$$

where the accuracy and stability of the integrator are determined by the choice of  $\alpha$ ,  $\beta$  and the timestep  $\Delta t$ . Figure 4(a) graphically illustrates the point  $\mathbf{x}$  and its displacement  $\mathbf{u}^n$  and current location  $\mathbf{y}^n = \mathbf{x} + \mathbf{u}^n$  at timestep  $n$ . We write (3.3) at timestep  $n + 1$  as

$$\mathbf{M} \ddot{\mathbf{u}}^{n+1} = \mathbf{f}^{n+1}(\mathbf{u}^{n+1}) + \mathbf{b}^{n+1}(\mathbf{u}^{n+1}). \quad (3.8)$$

Solving (3.6) for  $\ddot{\mathbf{u}}^{n+1}$  and substituting into (3.8) gives

$$\mathbf{M} \mathbf{u}^{n+1} = \beta (\Delta t)^2 [\mathbf{f}^{n+1}(\mathbf{u}^{n+1}) + \mathbf{b}^{n+1}(\mathbf{u}^{n+1})] + \mathbf{M} \left[ \mathbf{u}^n + \dot{\mathbf{u}}^n \Delta t + \left( \frac{1}{2} - \beta \right) (\Delta t)^2 \ddot{\mathbf{u}}^n \right]. \quad (3.9)$$

This is an implicit nonlinear equation for  $\mathbf{u}^{n+1}$ . We can rewrite (3.9) as the residual equation

$$\mathbf{r}(\mathbf{u}^{n+1}) = \mathbf{M} \mathbf{u}^{n+1} - \beta (\Delta t)^2 [\mathbf{f}^{n+1}(\mathbf{u}^{n+1}) + \mathbf{b}^{n+1}(\mathbf{u}^{n+1})] \quad (3.10)$$

$$- \mathbf{M} \left[ \mathbf{u}^n + \dot{\mathbf{u}}^n \Delta t + \left( \frac{1}{2} - \beta \right) (\Delta t)^2 \ddot{\mathbf{u}}^n \right], \quad (3.11)$$

and solve  $\mathbf{r}(\mathbf{u}^{n+1}) = \mathbf{0}$  for  $\mathbf{u}^{n+1}$  using Newton's method. Because the problem is in general nonlinear, the incremental solution strategy for a load step consists of finding an increment,  $\Delta \mathbf{u}$ , to the displacement  $\mathbf{u}^n$ . For Newton iteration  $k \geq 0$  of a load step we write

$$\mathbf{y}_{k+1}^{n+1} = \mathbf{y}_k^{n+1} + \Delta \mathbf{u}_k, \quad (3.12)$$

where

$$\mathbf{y}_k^{n+1} = \mathbf{x} + \mathbf{u}_k^{n+1}, \quad (3.13)$$

where  $\mathbf{u}_k^{n+1}$  is known from the previous iteration. A graphical illustration of the kinematics for iteration  $k$  is shown in Figure 4(b). Newton's method to determine  $\Delta \mathbf{u}_k$  requires a Jacobian matrix associated with the internal force. Using (3.3) and (3.13), the internal force density vector at time step  $n + 1$ , iteration  $k$ , is defined as

$$\mathbf{f}_k^{n+1}(\mathbf{u}_k^{n+1}) = \int_{\mathcal{H}_x} \{ \underline{\mathbf{T}}[\mathbf{x}, t^{n+1}] \langle \mathbf{x}' - \mathbf{x} \rangle - \underline{\mathbf{T}}[\mathbf{x}', t^{n+1}] \langle \mathbf{x} - \mathbf{x}' \rangle \} dV_{\mathbf{x}'}, \quad (3.14)$$

where the right-hand side of (3.14) is evaluated at  $\mathbf{u}_k^{n+1}$ . The first order approximation to  $\mathbf{f}_{k+1}^{n+1}$  is given as

$$\mathbf{f}_{k+1}^{n+1} \approx \mathbf{f}_k^{n+1} + \left. \frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{u}^{n+1}} \right|_{\mathbf{u}_k^{n+1}} \cdot \Delta \mathbf{u}_k, \quad (3.15)$$

where we define

$$\left. \frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{u}^{n+1}} \right|_{\mathbf{u}_k^{n+1}} \equiv \int_{\mathcal{H}_x} \{ (\underline{\mathbb{K}}[\mathbf{x}, t] \bullet \Delta \underline{\mathbf{U}}_k[\mathbf{x}, t]) \langle \mathbf{x}' - \mathbf{x} \rangle - (\underline{\mathbb{K}}[\mathbf{x}', t] \bullet \Delta \underline{\mathbf{U}}_k[\mathbf{x}', t]) \langle \mathbf{x} - \mathbf{x}' \rangle \} dV_{\mathbf{x}'}. \quad (3.16)$$

The rightmost term in (3.15) denotes the derivative of  $\mathbf{f}^{n+1}$  with respect to  $\mathbf{u}^{n+1}$  evaluated at  $\mathbf{u}_k^{n+1}$  in the direction  $\Delta \mathbf{u}_k$ . Important in the evaluation of this term is the modulus state  $\underline{\mathbb{K}}$  [20]. For implicit solution algorithms, evaluation of  $\underline{\mathbb{K}}$  is crucial for optimal convergence rates. For the elastic model of §2.3 and the elastic-plastic model of §2.4, analytical expressions for  $\underline{\mathbb{K}}$  are given in [14]. For the viscoelastic model of §2.5, an analytical expression for  $\underline{\mathbb{K}}$  is given in [13]. In Peridigm, the modulus state can also be evaluated numerically via a finite difference approximation, or exactly via the Trilinos automatic differentiation package Sacado [16].

The directional derivative of the residual (3.10) is the second-rank tensor

$$\left. \frac{\partial \mathbf{r}^{n+1}}{\partial \mathbf{u}^{n+1}} \right|_{\mathbf{u}_k^{n+1}} = \mathbf{M} - \beta(\Delta t)^2 \left( \frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{u}^{n+1}} + \frac{\partial \mathbf{b}^{n+1}}{\partial \mathbf{u}^{n+1}} \right) \Big|_{\mathbf{u}_k^{n+1}}. \quad (3.17)$$

Newton's method requires us to solve

$$\mathbf{r}(\mathbf{u}_{k+1}^{n+1}) \approx \mathbf{r}(\mathbf{u}_k^{n+1}) + \left. \frac{\partial \mathbf{r}}{\partial \mathbf{u}^{n+1}} \right|_{\mathbf{u}_k^{n+1}} \cdot \Delta \mathbf{u}_k = \mathbf{0} \quad (3.18)$$

for  $\Delta \mathbf{u}_k$ . For completeness, the residual required for each Newton iteration is given by

$$\begin{aligned} \mathbf{r}(\mathbf{u}_k^{n+1}) = & \mathbf{M} \mathbf{u}_k^{n+1} - \beta(\Delta t)^2 [\mathbf{f}_k^{n+1}(\mathbf{u}_k^{n+1}) + \mathbf{b}_k^{n+1}(\mathbf{u}_k^{n+1})] \\ & - \mathbf{M} \left[ \mathbf{u}^n + \dot{\mathbf{u}}^n \Delta t + \left( \frac{1}{2} - \beta \right) (\Delta t)^2 \ddot{\mathbf{u}}^n \right]. \end{aligned}$$

Note that values at step  $n$  do not include a subscript since they are converged values from the previous load step.

## Quasistatics

In a quasistatic simulation, (3.3) becomes

$$-\mathbf{f}(\mathbf{u}) = \mathbf{b}(\mathbf{u}), \quad (3.19)$$

and is discretized into a finite number fictitious time steps and written as

$$-\mathbf{f}^n(\mathbf{u}^n) = \mathbf{b}^n(\mathbf{u}^n), \quad (3.20)$$

where the time  $t$  here need not be the real time, but can refer to a parameter controlling the externally applied force  $\mathbf{b}$ . The proceeding equation is solved using the techniques of the previous section.

## 3.9 Contact

In the model discussed so far, elements interact only through their bond forces. To account for interactions between non-bonded elements, we utilize the simple contact model of [19]. We add to the force  $\mathbf{b}$  in (3.3), for each element  $i$ , the force

$$\mathbf{f}_S = \min \left\{ 0, \frac{c_S}{\delta} (\|\mathbf{y}_p - \mathbf{y}_i\| - d_{pi}) \right\} \frac{\mathbf{y}_p - \mathbf{y}_i}{\|\mathbf{y}_p - \mathbf{y}_i\|}, \quad (3.21)$$

where  $d_{pi}$  is a specified short-range interaction distance between elements  $p$  and  $i$ , and

$$c_S = \frac{18K}{\pi\delta^4}. \quad (3.22)$$

Note that the short-range force is always repulsive, never attractive. Contact forces appear only when elements are under compression.

## 3.10 Damage

Two elements in a body within each other's horizon in the reference configuration may stop interacting if damage is incurred. This manifests in a simulation by removing these elements from each other's bond family  $\mathcal{F}_i$  (see (3.1)). Peridigm supports the *critical stretch* bond breaking criteria, as discussed in §2.6. When instantiating a damage model in the input deck, a user must input a critical stretch.

## 3.11 Output

Peridigm supports the ExodusII output format [24] through the Trilinos package SEACAS [24]. Although formally known as ExodusII, we will refer to this format simply as the Exodus format in the following. Exodus is based upon NetCDF, and also includes the nemesis parallel extensions. Exodus is fundamentally a serial output format, and each MPI process will open and write its own

Exodus database for all the elements that it owns. If a load rebalance occurs during the course of a parallel simulation, each MPI process must close its Exodus database and open a new database containing the elements owned by that process after the rebalance. The net effect is that many Exodus output files may be generated during the course of a parallel simulation. Tools in the SEACAS package to help in merging these separate files together into a single Exodus database. To automate this process, a python script is distributed with Peridigm, and can be executed as

```
MergeFiles.py <File Base Name> <# of Processors>
```

where we are executing the MergeFiles.py script from the Peridigm install directory, not the source directory. This script will produce a single Exodus database file containing the complete simulation output.

### 3.12 Visualization

Any visualization package capable of reading and displaying Exodus-format data may be used to visualize the output of a Peridigm simulation. We use Paraview [12], an open-source, multi-platform data analysis and visualization application. Binary installers for Windows, Linux, and MacOS are distributed from the Paraview website.

### 3.13 Pitfalls

**Defining the peridynamic horizon  $\delta$ .** The user must specify the horizon  $\delta$  in the input deck. This argument determines which elements interact when the simulation is initialized. It is recommended that  $\delta$  be set to a small fraction of a lattice constant larger than desired.

For example, if the mesh spacing is 0.0005 and you wish to set the horizon to three times the mesh spacing, then set  $\delta$  to be 0.0015001, a value slightly larger than three times the mesh spacing. This guarantees that elements three lattice constants away from each other are still bonded. If  $\delta$  is set to 0.0015, for example, floating point error may result in some pairs of elements three mesh spacings apart not being bonded.

### 3.14 Bugs

The user is cautioned that this code is under active development. If you are confident that you have found a bug, please send an email to the developers. The most useful thing you can do for us is to isolate the problem. Run it on the smallest problem and fewest number of processors and with the simplest input script that reproduces the bug. In your email, please describe the problem and any ideas you have as to what is causing it or where in the code the problem might be. We'll request your input script and data files if necessary.

## 4 Extending Peridigm

Peridigm was developed to be easily user extensible. In particular, there are two easy mechanisms to extend Peridigm for your own purposes. The first, compute classes, allow a user to write only serial code to compute any quantity of interest as a function of any state variable in Peridigm. The second, material model classes, allow a user to easily implement their own material model. Users are requested to produce unit, regression, or verification tests for any new features deployed into the Peridigm distribution.

### 4.1 Compute Classes

Compute classes, found in `src/compute`, allow users to compute any quantity of interest. Any compute class is required to implement the following interface:

```
1: //! Default constructor
2: Compute()
3:
4: //! Destructor
5: virtual Compute()
6:
7: //! Returns the fieldspecs computed by this class
8: virtual std::vector<Field_NS::FieldSpec> getFieldSpecs() const = 0;
9:
10: //! Initialize the compute class
11: virtual void initialize( Teuchos::RCP< std::vector<PeridigmNS::Block> > blocks ) const ;
12:
13: //! Perform computation
14: virtual int compute( Teuchos::RCP< std::vector<PeridigmNS::Block> > blocks ) const = 0;
```

The `getFieldSpecs()` method returns an `std::vector` of `FieldSpecs`, telling Peridigm what data it should allocate for the compute class. A `fieldspec` can be thought of as a data type used by Peridigm. A complete listing of all `fieldspecs` can be found in `src/io/mesh_output/Field.h`, and includes scalar and vector quantities defined for nodes and elements as well as global scalar and vector quantities.

The `initialize()` method allows the compute class an opportunity to initialize any storage or private state data before the simulation begins. Lastly, the `compute()` method computes the quantity of interest.

After creating a new compute class, be sure to add it to the file `src/compute/compute_includes.hpp`. After that, re-run `cmake` in your Peridigm build directory, and then rebuild Peridigm. Your compute class can now be requested as an output field in any Peridigm input deck.

As a practical example, we walk through the `Peridigm_Compute_Acceleration` compute class. The source for the relevant methods of the compute class appears below:

```
1: std::vector<Field_NS::FieldSpec> PeridigmNS::Compute_Acceleration::getFieldSpecs() const {
2: std::vector<Field_NS::FieldSpec> myFieldSpecs;
3: myFieldSpecs.push_back(Field_NS::ACCEL3D);
```

```

4: return myFieldSpecs;
5: }
6:
7: int PeridigmNS::Compute_Acceleration::compute( Teuchos::RCP< std::vector<PeridigmNS::Block> > blocks
  ) const {
8: int retval(0);
9:
10: Teuchos::RCP<Epetra_Vector> force, acceleration;
11: std::vector<PeridigmNS::Block>::iterator blockIt;
12: for(blockIt = blocks->begin() ; blockIt != blocks->end() ; blockIt++){
13:   Teuchos::RCP<PeridigmNS::DataManager> dataManager = blockIt->getDataManager();
14:   force = dataManager->getData(Field_NS::FORCE_DENSITY3D, Field_ENUM::STEP_NP1);
15:   acceleration = dataManager->getData(Field_NS::ACCEL3D, Field_ENUM::STEP_NP1);
16:   *acceleration = *force;
17:   double density = blockIt->getMaterialModel()->Density();
18:   // Report if any calls to Scale() failed.
19:   retval = retval || acceleration->Scale(1.0/density);
20: }
21: return retval;
22: }

```

The `getFieldSpecs()` routine returns the fieldspec for a nodal acceleration vector, telling Peridigm to ensure storage is allocated for that vector field variable. The `compute()` routine takes an `std::vector` of material blocks and, for each block, extracts the force and acceleration vectors for the nodes in that block. It initially fills the acceleration vector by copying the force vector, and then divides by the density.

The `Compute_Acceleration` routine is invoked whenever the acceleration is requested as an output. For example, in the Peridigm input deck in Algorithm 2, line 85, the acceleration is requested as one of several outputs. The acceleration compute class above is called every time output is written to disk.

After creating a new compute class, just specify that compute class in the requested output fields and it will be called every time output is written to disk.

## 4.2 Material Models

Material model classes, found in `src/materials`, allow users to compute any quantity of interest. Any compute class is required to implement the following interface:

```

1: //! Standard constructor
2: Material(const Teuchos::ParameterList & params) {}
3:
4: //! Destructor
5: virtual Material(){}
6:
7: //! Return name of material type
8: virtual string Name() const = 0;
9:
10: //! Returns the density of the material
11: virtual double Density() const = 0;
12:

```

```

13: //! Returns the bulk modulus of the material
14: virtual double BulkModulus() const = 0;
15:
16: //! Returns the shear modulus of the material
17: virtual double ShearModulus() const = 0;
18:
19: //! Returns the horizon
20: virtual double Horizon() const = 0;
21:
22: //! Returns a vector of field specs that specify the variables associated with the material
23: virtual Teuchos::RCP< std::vector<Field_NS::FieldSpec> > VariableSpecs() const = 0;
24:
25: //! Initialize the material model
26: virtual void initialize( ...) const {}
27:
28: //! Evaluate the internal force
29: virtual void computeForce( ...) = 0;
30:
31: //! Evaluate the jacobian
32: virtual void computeJacobian( ...) const;

```

Most of the methods return basic material properties. The `VariableSpecs()` method returns an `std::vector` of `FieldSpecs`, telling Peridigm what data it should allocate for the material model class. The `initialize` method initializes any allocated storage, and the `computeForce` method fills the force vector as a function of the current state data (positions, displacements, velocity, etc.) For implicit time integration and quasistatics, the `computeJacobian` method is required.

### 4.3 Creating Tests

Any new compute classes or material models added to Peridigm should be supported by unit, regression, or verification tests. Most source directories have `unit_test` directories that contain unit tests for code in that directory. The `test/verification` and `test/regression` directories contain verification and regression tests, respectively. Any of these tests can be used as a template for the generation of new tests.

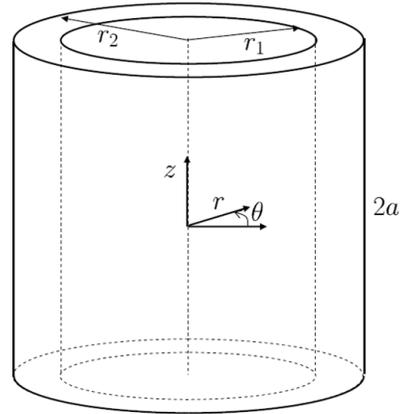
## 5 A Numerical Example

To introduce Peridigm, we work through a nontrivial example problem, describing the setup, simulation, and visualization steps.

### 5.1 Problem Description and Setup

Motivated by the tube fragmentation experiments of Winter [26] and Vogler [25] as explained by Grady [10], we consider the fragmentation of a homogeneous hollow cylinder. The geometry of the cylinder and its material properties are shown in Figure 5.

Property	Value
Inner Radius $r_1$	0.020 m
Outer Radius $r_2$	0.025 m
Length $2a$	0.100 m
Density $\rho$	8700 kg/m <sup>3</sup>
Bulk modulus $K$	130 GPa
Shear modulus $\mu$	78 GPa
Yield stress $Y$	500 GPa
Ultimate stress $Y$	700 GPa
Elongation at failure $Y$	0.02



**Figure 5.** Cylinder geometry and material properties.

The initial velocity of the cylinder is prescribed to be

$$v(r) = V_{r0}V_{r1} \left(\frac{a}{z}\right)^2, \quad (5.1)$$

$$v(z) = V_{z0} \left(\frac{a}{z}\right), \quad (5.2)$$

$$v(\theta) = 0, \quad (5.3)$$

where  $V_{r0} = 200$  m/s,  $V_{r1} = 50$  m/s,  $V_{z0} = 100$  m/s. To model a brittle cylinder, we assume the linear peridynamic solid model of §2.3.

### 5.2 Writing the Peridigm Input File

We discuss the example input script from Algorithm 2, covering the sections discussed in §3.2 that are used in this example. This input script is distributed with Peridigm as `examples/fragmenting_cylinder/fragmenting_cylinder.xml`.

The discretization section begins on line 5. PdQuickGrid, Peridigm’s internal mesh generator, is used. The horizon is specified, and the `Spherical` neighborhood type specifies that the distances between points are to be computed using a 2-norm. Line 9 begins the section that specifies the geometry of the cylinder. In particular, line 17 specifies the number of mesh elements through the thickness of the cylinder.

Line 21 starts the material model section. In this section, all material models used in the simulation must be described and material properties given. In this example, we use only the linear peridynamic solid model of §2.3.

Line 30 starts the damage models section. In this example, we use the critical stretch damage model of §2.6.

Line 37 starts the material blocks section. In this section, each material block must be associated with a material model, and, optionally, a damage model. User-specified names (i.e., “My Linear Elastic Material”) are associated with each block by name. In this example, there is only one material block.

Line 45 starts the section where initial and boundary conditions are specified. In this example, we specify an initial velocity for every element in the cylinder as a function of its  $(x, y, z)$  coordinates.

Line 66 starts the solver section, where the solver and its parameters are specified. In this example, we do explicit time integration using the integrator described in §3.7.

Line 75 starts the output section, where the output file type and output variables are specified as well as the output filename and frequency of output. Every variable listed on the output variables section on line 81 will be output.

### 5.3 Running Peridigm and Visualizing Output

For expediency, we run this example in parallel as

```
mpirun -np 12 Peridigm fragmenting_cylinder.xml
```

When the progress bar has reached 100%, the simulation is complete. As Exodus is a fundamentally serial output format, each MPI process writes its own Exodus database. For ease in visualizing the entire simulation, it is convenient to merge the separate databases into a single database. A python script is distributed with Peridigm to automate this process, and can be executed as

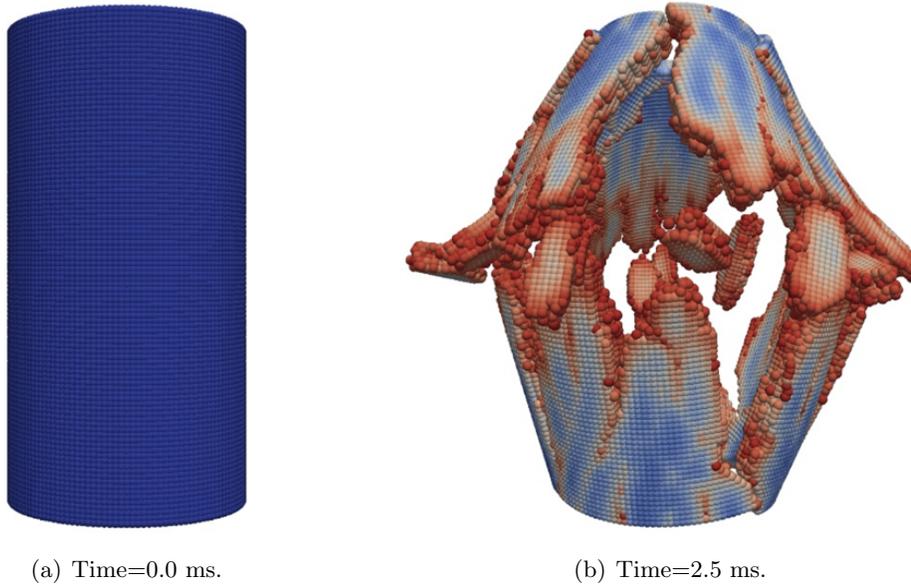
```
MergeFiles.py fragmenting_cylinder 12
```

where we are executing the MergeFiles.py script from the Peridigm install directory, not the source directory. This script will produce a single exodus database file `fragmenting_cylinder.e` containing the complete simulation output.

To visualize the results, run

Paraview fragmenting\_cylinder.e

Be sure to hit the green “apply” button on the right. Press the “play” button at the top to display the simulation results. One can use the “glyph” filter in Paraview to replace each element by a small sphere, and then color those spheres by the element damage. The results are shown in Figure 5.3.



**Figure 6.** Brittle cylinder before and after simulation. Color indicates damage (red = damaged, blue = undamaged).

---

**Algorithm 2** Peridigm Input Deck for Fragmenting Brittle Cylinder

---

```
1: <ParameterList>
2:
3:   <Parameter name="Verbose" type="bool" value="false"/>
4:
5:   <ParameterList name="Discretization">
6:     <Parameter name="Type" type="string" value="PdQuickGrid"/>
7:     <Parameter name="Horizon" type="double" value="0.00417462"/>
8:     <Parameter name="NeighborhoodType" type="string" value="Spherical"/>
9:     <ParameterList name="TensorProductCylinderMeshGenerator">
10:      <Parameter name="Type" type="string" value="PdQuickGrid"/>
11:      <Parameter name="Inner Radius" type="double" value="0.020"/>
12:      <Parameter name="Outer Radius" type="double" value="0.025"/>
13:      <Parameter name="Cylinder Length" type="double" value="0.100"/>
14:      <Parameter name="Ring Center x" type="double" value="0.0"/>
15:      <Parameter name="Ring Center y" type="double" value="0.0"/>
16:      <Parameter name="Z Origin" type="double" value="0.0"/>
17:      <Parameter name="Number Points Radius" type="int" value="5"/>
18:    </ParameterList>
19:  </ParameterList>
20:
21:  <ParameterList name="Materials">
22:    <ParameterList name="My Linear Elastic Material">
23:      <Parameter name="Material Model" type="string" value="Elastic"/>
24:      <Parameter name="Density" type="double" value="7800.0"/>
25:      <Parameter name="Bulk Modulus" type="double" value="130.0e9"/>
26:      <Parameter name="Shear Modulus" type="double" value="78.0e9"/>
27:    </ParameterList>
28:  </ParameterList>
29:
30:  <ParameterList name="Damage Models">
31:    <ParameterList name="My Critical Stretch Damage Model">
32:      <Parameter name="Damage Model" type="string" value="Critical Stretch"/>
33:      <Parameter name="Critical Stretch" type="double" value="0.02"/>
34:    </ParameterList>
35:  </ParameterList>
36:
37:  <ParameterList name="Blocks">
38:    <ParameterList name="My Group of Blocks">
39:      <Parameter name="Block Names" type="string" value="block_1"/>
40:      <Parameter name="Material" type="string" value="My Linear Elastic Material"/>
41:      <Parameter name="Damage Model" type="string" value="My Critical Stretch Damage Model"/>
42:    </ParameterList>
43:  </ParameterList>
44:
```

---

---

```

45: <ParameterList name="Boundary Conditions">
46:   <ParameterList name="Initial Velocity X">
47:     <Parameter name="Type" type="string" value="Initial Velocity"/>
48:     <Parameter name="Node Set" type="string" value="All"/>
49:     <Parameter name="Coordinate" type="string" value="x"/>
50:     <Parameter name="Value" type="string" value="(200-50*((z/0.05)-1)^2)*cos(atan2(y,x))+rnd(5)"/>
51:   </ParameterList>
52:   <ParameterList name="Initial Velocity Y">
53:     <Parameter name="Type" type="string" value="Initial Velocity"/>
54:     <Parameter name="Node Set" type="string" value="All"/>
55:     <Parameter name="Coordinate" type="string" value="y"/>
56:     <Parameter name="Value" type="string" value="(200-50*((z/0.05)-1)^2)*sin(atan2(y,x))+rnd(5)"/>
57:   </ParameterList>
58:   <ParameterList name="Initial Velocity Z">
59:     <Parameter name="Type" type="string" value="Initial Velocity"/>
60:     <Parameter name="Node Set" type="string" value="All"/>
61:     <Parameter name="Coordinate" type="string" value="z"/>
62:     <Parameter name="Value" type="string" value="(100*((z/0.05)-1))+rnd(5)"/>
63:   </ParameterList>
64: </ParameterList>
65:
66: <ParameterList name="Solver">
67:   <Parameter name="Verbose" type="bool" value="false"/>
68:   <Parameter name="Initial Time" type="double" value="0.0"/>
69:   <Parameter name="Final Time" type="double" value="2.5e-4"/>
70:   <ParameterList name="Verlet">
71:     <Parameter name="Fixed dt" type="double" value="1.0e-8"/>
72:   </ParameterList>
73: </ParameterList>
74:
75: <ParameterList name="Output">
76:   <Parameter name="Output File Type" type="string" value="ExodusII"/>
77:   <Parameter name="Output Format" type="string" value="BINARY"/>
78:   <Parameter name="Output Filename" type="string" value="fragmenting_cylinder"/>
79:   <Parameter name="Output Frequency" type="int" value="250"/>
80:   <Parameter name="Parallel Write" type="bool" value="true"/>
81:   <ParameterList name="Output Variables">
82:     <Parameter name="Proc_Num" type="bool" value="true"/>
83:     <Parameter name="Displacement" type="bool" value="true"/>
84:     <Parameter name="Velocity" type="bool" value="true"/>
85:     <Parameter name="Acceleration" type="bool" value="true"/>
86:     <Parameter name="Force_Density" type="bool" value="true"/>
87:     <Parameter name="ID" type="bool" value="true"/>
88:     <Parameter name="Dilatation" type="bool" value="true"/>
89:     <Parameter name="Damage" type="bool" value="true"/>
90:     <Parameter name="Weighted_Volume" type="bool" value="true"/>
91:   </ParameterList>
92: </ParameterList>
93:
94: </ParameterList>

```

---

## References

- [1] *CMake - Cross Platform Make*. <http://www.cmake.org/>.
- [2] *The CUBIT Tool Suite Web Page*. <http://cubit.sandia.gov>.
- [3] *DAKOTA Project Web Page*. <http://dakota.sandia.gov>.
- [4] *Trilinos project web page*. <http://trilinos.sandia.gov>.
- [5] *Trilinos tutorial*, Tech. Report SAND2004-2189, Sandia National Laboratories, 2004. Available at <http://trilinos.sandia.gov/Trilinos10.12Tutorial.pdf>.
- [6] B. ADAMS, W. BOHNHOFF, K. DALBEY, J. EDDY, M. ELDRED, D. GAY, K. HASKELL, P. HOUGH, AND L. SWILER, *Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.0 user's manual*, Tech. Report SAND2010-2183, Sandia National Laboratories, 2009.
- [7] T. BELYTSCHKO, W. K. LIU, AND B. MORAN, *Nonlinear Finite Elements for Continua and Structures*, Wiley, 2000.
- [8] I. BERG, *MuParser webpage*. <http://muparser.sourceforge.net/index.html>.
- [9] J. T. FOSTER, *Dynamic crack initiation toughness: Experiments and peridynamic modeling*, Tech. Report SAND2009-7217, Sandia National Laboratories, 2009.
- [10] D. GRADY, *Fragmentation of Rings And Shells: The Legacy of N.F. Mott*, Springer, 2006.
- [11] M. HEROUX, R. BARTLETT, R. H. VICKI HOWLE, J. HU, T. KOLDA, R. LEHOUCQ, K. LONG, R. PAWLOWSKI, E. PHIPPS, A. SALINGER, H. THORNQUIST, R. TUMINARO, J. WILLENBRING, AND A. WILLIAMS, *An overview of trilinos*, Tech. Report SAND2003-2927, Sandia National Laboratories, 2003.
- [12] KITWARE INC., *ParaView web page*. <http://www.paraview.org/>.
- [13] J. A. MITCHELL, *A non-local, ordinary-state-based viscoelasticity model for peridynamics*, Tech. Report SAND2011-8064, Sandia National Laboratories, 2011.
- [14] —, *A nonlocal, ordinary, state-based plasticity model for peridynamics*, Tech. Report SAND2011-3166, Sandia National Laboratories, 2011.
- [15] R. PAWLOWSKI AND E. PHIPPS, *Phalanx webpage*. <http://trilinos.sandia.gov/packages/phalanx/>.
- [16] E. PHIPPS, *Sacado webpage*. <http://trilinos.sandia.gov/packages/sacado/>.
- [17] P. SELESON AND M. PARKS, *On the role of the influence function in the peridynamic theory*, Int. J. Mult. Comp. Eng., (2010). Submitted.
- [18] S. A. SILLING, *Reformulation of elasticity theory for discontinuities and long-range forces*, Journal of the Mechanics and Physics of Solids, 48 (2000), pp. 175–209.

- [19] S. A. SILLING. Personal communication, 2007.
- [20] S. A. SILLING, *Linearized theory of peridynamic states*, Journal of Elasticity, 99 (2010), pp. 85–111.
- [21] S. A. SILLING AND E. ASKARI, *A meshfree method based on the peridynamic model of solid mechanics*, Computer and Structures, 83 (2005), pp. 1526–1535.
- [22] S. A. SILLING, M. EPTON, O. WECKNER, J. XU, AND E. ASKARI, *Peridynamic states and constitutive modeling*, J. Elasticity, 88 (2007), pp. 151–184.
- [23] S. A. SILLING AND R. B. LEHOUCQ, *Peridynamic theory of solid mechanics*, Advances in Applied Mechanics, 44 (2010), pp. 73–168.
- [24] G. D. SJAARDEMA, L. A. SCHOOF, AND V. R. YARBERRY, *Exodus ii: A finite element data model*, Tech. Report SAND92-2137, Sandia National Laboratories, 2006.
- [25] T. VOGLER, T. THORNHILL, W. REINHART, L. CHHABILDAS, D. GRADY, L. WILSON, O. HURRICANE, AND A. SUNWOO, *Fragmentation of materials in expanding tube experiments*, Int. J. Impact Eng., 29 (2003), pp. 735–746.
- [26] R. WINTER, *Measurement of fracture strain at high strain rates*, Inst. Phys. Conf. Ser., 47 (1979), pp. 81–89.

## DISTRIBUTION:

1	MS 1322	John Aidun, 1425
1	MS 1318	Steve Bond, 1444
1	MS 1320	Richard Lehoucq, 1444
1	MS 1320	John Mitchell, 1444
1	MS 1320	Michael Parks, 1444
1	MS 1320	Stewart Silling, 1444
1	MS 1320	Randy Summers, 1444
1	MS 0899	Technical Library, 9536 (electronic copy)







**Sandia National Laboratories**