AEE 500
Peridynamic Modelling of Materials and Fracture

Erhan Haliloğlu Celiloğlu
2017648

May 22, 2023

1 Introduction

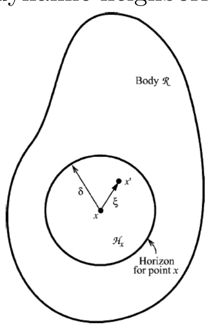
Peridynamics is a non-local continuum theory where discontinuities within a continua are handled with integral equtions. The integration consists of a domain of influence which is typically a spherical domain with a radius, δ, where the quantity of interest is integrated (averaged) over the neighborhood of the location of interest. Considering balance of linear momentum in both local and non-local formulations, following relations show differentitation-to-integration change in both theories;

$$\rho \dot{\mathbf{v}} = \frac{\partial \sigma}{\partial x} + \rho \mathbf{b} \tag{1}$$

$$\rho \dot{\mathbf{v}} = \int_{H_x} \mathbf{f} \, dV + \mathbf{b} \tag{2}$$

(1) is known as the *Cauchy's first law of motion* where $\sigma(\mathbf{x}, t)$ is the Cauchy stress tensor whereas (2) is the *Peridynamic equation of motion* where $\mathbf{f}$ if the Peridynamic force density vector, ρ is the mass density and $\mathbf{b}$ is the body force density. The integration domain is set by a parameter δ which defines a domain of influence or neighborhood at a point x.

Figure 1: Peridynamic neighborhood in a body



The integration over the domain of influence is usually used in proofs or derivations whereas when a body is discritized into points, summation over a neighborhood is preferred;

$$\rho_k \dot{\mathbf{v}}_k = \sum_{j=1}^{N_k} \mathbf{f}(x_k, x_j, ..., t) V_j + \mathbf{b}_k \tag{3}$$

In the following chapters, construction of peridynamic force vector and vector states, constitutive models and solution techniques are discussed.

2 Vector States

Preliminary Identities: Vector states or *Peridynamic States* are mappings from pairs of points to some quantities.[Silling and Lehoucq(2010)] A peridynamic state $A\langle\bullet\rangle$ is a function over a vector. Depending on return value it maybe scalar state, vector state or double state which returns second - order tensors.

Some of the identities of states are given below;

$$Identitiy\, State\ : \underline{X}\langle\vec{\xi}\rangle = \vec{\xi} \tag{4}$$

$$Null\, State\ : \underline{0}\langle\vec{\xi}\rangle = \vec{0} \tag{5}$$

$$Dot\, Product\ : \underline{A} \bullet \underline{B} = \int_H \underline{A}\langle\vec{\xi}\rangle \cdot \underline{B}\langle\vec{\xi}\rangle \, dV_\xi \tag{6}$$

$$Norm\ : \|\underline{A}\| = \sqrt{\underline{A} \bullet \underline{A}} \tag{7}$$

Fretchet Derivative of Functions of States: Define a potential function,$\psi(\bullet)$, which is a function of a scalar state, its *Fretchet derivative*,$\nabla\psi$, is defined such that;

$$\psi(\underline{A} + \underline{a}) = \psi(\underline{A}) + \nabla\psi(\underline{A}) \cdot \underline{a} + o(\|\underline{a}\|) \tag{8}$$

3 Constitutive Models

Deformation State: Constitutive model in peridynamics provides the internal force vector state as a function of deformation vector state. Deformation state, $\underline{Y}[\vec{x}, t]$, maps the relative distances (bonds) in reference configurations to deformed images such that;

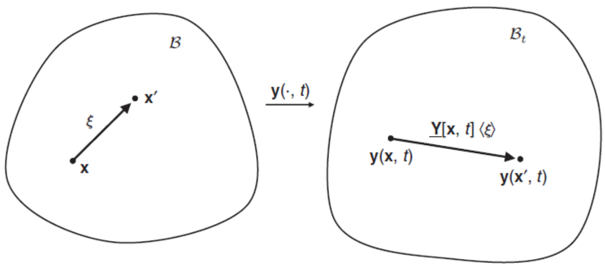
$$\underline{Y}[\vec{x}, t]\langle\vec{\xi}\rangle = \vec{y}(\vec{x}', t) - y(\vec{x}, t) \tag{9}$$

The vectors that denote the relative positions or bond vectors in reference and deformed configurations are denoted as ;

$$\vec{\xi} = \vec{x'} - \vec{x} \tag{10}$$

$$\vec{\eta} = \vec{y'} - \vec{y} \tag{11}$$

Figure 2: Deformation state as a mapping



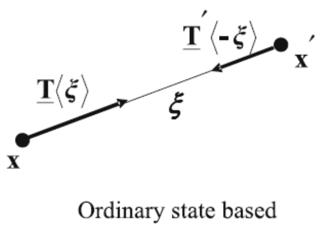
Force State: Internal forces within the continuum are defined by the counteracting force vectors in ordinary state based theory. Force vector state, then, is a mapping over the bond vector $\vec{\xi}$ such that;

$$\vec{t}(\vec{x}, \vec{x}', t) = \underline{T}[\vec{x}, t]\langle\vec{\xi}\rangle \tag{12}$$

Force state is called ordinary if following holds,

$$\underline{T} = \underline{\hat{T}} = t\underline{M} \tag{13}$$

Figure 3: Ordinary state based force vectors



$$\underline{M} = \frac{\underline{Y}}{|\underline{Y}|} \tag{14}$$

If the material is elastic, free energy density only depends on $\underline{Y}$ such that,

$$\underline{\hat{T}} = \nabla\hat{W} \tag{15}$$

where $\nabla\hat{W}$ is the Fretchet derivative of free energy density.

In ordinary state based peridynamics, peridynamic equation of motion 2 takes the form of;

$$\rho\dot{\mathbf{v}} = \int_V (\mathbf{t} - \mathbf{t'})\, dV + \rho\mathbf{b} \tag{16}$$

where,

$$\vec{t}(\vec{x}, \vec{x}', t) = \underline{T}\langle\vec{\xi}\rangle = \underline{T}\langle\vec{x'} - \vec{x}\rangle \tag{17}$$

$$\vec{t'}(\vec{x}, \vec{x}', t) = \underline{T}\langle\vec{\xi'}\rangle = \underline{T}\langle\vec{x} - \vec{x'}\rangle \tag{18}$$

Linear Elastic Solids: Suppose the free energy density of a material is given as,

$$W(\theta, \underline{e}^d) = \frac{\kappa\theta^2}{2} + \frac{\alpha}{2}(\underline{\omega}\underline{e}^d) \cdot \underline{e}^d \tag{19}$$

where κ is the bulk modulus, θ is the volumetric strain (dilatation), α is the shear modulus, $\underline{\omega}$ is the influence scalar state, $\underline{e}^d$ is the distortional (deviatoric) part of the extension scalar state such that;

$$\underline{x} = |\underline{X}| \tag{20}$$

$$\underline{e} = |\underline{Y}| - \underline{x} \tag{21}$$

$$\theta = 3\frac{(\underline{\omega}\underline{x}) \cdot \underline{e}}{(\underline{\omega}\underline{x}) \cdot \underline{x}} \tag{22}$$

$$\underline{e}^d = \underline{e} - \frac{\theta\underline{x}}{3} \tag{23}$$

These relations results in force vector state as

$$t\vec{M} = \left(\frac{3}{m}\kappa\theta\underline{\omega}|\vec{\xi}| + \frac{15\mu}{m}\underline{\omega}\underline{e}^d\right)\frac{\vec{\eta} + \vec{\xi}}{|\vec{\eta} + \vec{\xi}|} \tag{24}$$

As oppose to these relations, [Madenci and Oterkus(2013)] made use of different formulations based on similar decompositions over ordinary state based peridynamics theory. According to [Madenci and Oterkus(2013)], strain energy density and dilatation of a material is given as;

$$W_{(k)} = a\theta_{(k)}^2 + \sum_{j=1}^{N} b\omega_{(k)(j)}((|\vec{\xi}_{(k)(j)} + \vec{\eta}_{(k)(j)}| - |\vec{\xi}_{(k)(j)}|)V_{(j)} \tag{25}$$

$$\theta_{(k)} = d\sum_{j=1}^{N} \frac{|\vec{\xi}_{(k)(j)} + \vec{\eta}_{(k)(j)}|}{|\vec{\xi}_{(k)(j)}|}\frac{\vec{\xi}_{(k)(j)} + \vec{\eta}_{(k)(j)}}{|\vec{\xi}_{(k)(j)} + \vec{\eta}_{(k)(j)}|} \cdot \vec{\xi}_{(k)(j)}V_{(j)} \tag{26}$$

where the subscript k denotes the hosting material point of a neighborhood in a body and subscript j denotes the j^{th} neighbor of material point $x_{(k)}$. Note that the integrations over the domain are transformed into summations.

This volumetric and distorsional decomposition of the strain energy density is reflected over the force vectors such that,

$$t_{(k)(j)} = \frac{1}{2}A\frac{\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}}{|\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}|} \tag{27}$$

$$t_{(j)(k)} = -\frac{1}{2}B\frac{\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}}{|\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}|} \tag{28}$$

$$A = 4\omega_{(k)(j)}\left(d\frac{\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}}{|\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}|} \cdot \frac{\vec{\xi}_{(k)(j)}}{|\vec{\xi}_{(k)(j)}|}a\theta_{(k)} + b\left(|\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}| - |\vec{\xi}_{(k)(j)}|\right)\right) \tag{29}$$

$$B = 4\omega_{(j)(k)}\left(d\frac{\vec{\eta}_{(j)(k)} + \vec{\xi}_{(j)(k)}}{|\vec{\eta}_{(j)(k)} + \vec{\xi}_{(j)(k)}|} \cdot \frac{\vec{\xi}_{(j)(k)}}{|\vec{\xi}_{(j)(k)}|}a\theta_{(j)} + b\left(|\vec{\eta}_{(j)(k)} + \vec{\xi}_{(j)(k)}| - |\vec{\xi}_{(j)(k)}|\right)\right) \tag{30}$$

where, a,d and b are called peridynamic parameters which are yet to be determined based on material properties. For an isotropic material model in three dimensions, these parameters are resolved into;

$$a = \frac{1}{2}(\kappa - \frac{5\mu}{3}) \tag{31}$$

$$b = \frac{15\mu}{2\pi\delta^5} \tag{32}$$

$$d = \frac{9}{4\pi\delta^4} \tag{33}$$

The influence scalar state $\omega_{(k)(j)}$ is also given as;

$$\omega_{(k)(j)} = \frac{\delta}{|\xi_{(k)(j)}|} \tag{34}$$

Damage Models: [Silling and Askari(2005)] shows that a critical threshold value for a bond can be determined such that if it strecthes beyond that it maybe eliminated. The relation to Grifith's fracture theory as follows for bond based models,

$$s_c = \sqrt{\frac{5G_0}{9\kappa\delta}} \tag{35}$$

where, G_0 is the Critical Energy Release Rate. [Madenci and Oterkus(2013)] also utilizes a similar threshold value deriving for ordinary state based models,

$$s_c = \begin{cases} \sqrt{\frac{G_c}{(3\mu + (\frac{3}{4}^4)(\kappa - \frac{5\mu}{3}))\delta}} & ,3-D \\ \sqrt{\frac{G_c}{(\frac{6}{\pi}\mu + \frac{16}{9\pi^2}(\kappa - 2\mu))\delta}} & ,2-D \end{cases}$$

When the critical strecth value is reached, the bond must be eliminated. So, a step function $\mu(s_{(k)(j)}, t)$ is embedded in 26,29 and 30 such that;

$$\theta_{(k)} = d\sum_{j=1}^{N} \mu_{(k)(j)} \frac{|\vec{\xi}_{(k)(j)} + \vec{\eta}_{(k)(j)}|}{|\vec{\xi}_{(k)(j)}|} \frac{\vec{\xi}_{(k)(j)} + \vec{\eta}_{(k)(j)}}{|\vec{\xi}_{(k)(j)} + \vec{\eta}_{(k)(j)}|} \cdot \vec{\xi}_{(k)(j)} V_{(j)} \tag{36}$$

$$A = 4\omega_{(k)(j)} \left((d \frac{\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}}{|\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}|} \cdot \frac{\vec{\xi}_{(k)(j)}}{|\vec{\xi}_{(k)(j)}|} a\theta_{(k)} + b\mu_{(k)(j)} \left(|\vec{\eta}_{(k)(j)} + \vec{\xi}_{(k)(j)}| - |\vec{\xi}_{(k)(j)}| \right) \right) \tag{37}$$

$$B = 4\omega_{(j)(k)} \left(d \frac{\vec{\eta}_{(j)(k)} + \vec{\xi}_{(j)(k)}}{|\vec{\eta}_{(j)(k)} + \vec{\xi}_{(j)(k)}|} \cdot \frac{\vec{\xi}_{(j)(k)}}{|\vec{\xi}_{(j)(k)}|} a\theta_{(j)} + b\mu_{(k)(j)} \left(|\vec{\eta}_{(j)(k)} + \vec{\xi}_{(j)(k)}| - |\vec{\xi}_{(j)(k)}| \right) \right) \tag{38}$$

As the damage progresses in a body, a local damage index notation maybe used to quantify the progression of a crack at a material point $x_{(k)}$ such that;

$$\varphi(\vec{x_{(k)}}, t) = 1 - \frac{\int_H \mu(\vec{\xi}, t) dV}{\int_H dV} = 1 - \frac{\sum_{j=1}^{N_{(k)}} \mu_{(k)(j)} dV_{(j)}}{\sum_{j=1}^{N_{(k)}} dV_{(j)}} \tag{39}$$

Time Integration: As suggested in [Littlewood(2015)] explicit time integration with central difference scheme maybe utilized for a transient analysis with peridynamic formulations.

Scheme requires a midstep, $t^{n+\frac{1}{2}}$, update on the velocity and displacement of each point. Following to this update, internal forces are computed utilizing the peridynamic constitutive model and resulting acceleration of each point is calculated. Second velocity update is then done using updated velocity and accelerations.

Algorithm is as follows;

1. Initalize: $n = 0$,$t = 0$,$\vec{u} = \vec{0}$,$\vec{a} = \vec{0}$. Set initial conditions,
2. Set: Δt,
3. Calculate: $t^{n+1} = t^n + \Delta t$, $t^{n+\frac{1}{2}} = \frac{1}{2}(t^n + t^{n+1})$,
4. Update: $\vec{v}^{n+\frac{1}{2}} = \vec{v}^n + (t^{n+\frac{1}{2}} - t^n)\vec{a}^n$,
5. Update: $\vec{u}^{n+1} = \vec{u}^n + \vec{v}^{n+\frac{1}{2}}\Delta t$,
6. Calculate: $\vec{f}^{n+1}_{int} = \sum_{j=1}^{N} \left(\vec{t}(\vec{\xi}, \vec{\eta}^{n+1}) - \vec{t}'(\vec{\xi}, \vec{\eta}^{n+1})\right) V_j$,
7. Calculate: $\vec{f}^{n+1} = \vec{f}^{n+1}_{int} + \vec{f}^{n+1}_{ext}$, $\vec{a}^{n+1} = \mathbf{M}^{-1}\vec{f}^{n+1}$,
8. Update: $\vec{v}^{n+1} = \vec{v}^{n+\frac{1}{2}} + (t^{n+1} - t^{n+\frac{1}{2}})\vec{a}^{n+1}$,
9. Go to 3.

Since explicit time integration methods are conditionally stable, a stable time step must be set. [Silling and Askari(2005)] showed that such a time step maybe calculated with;

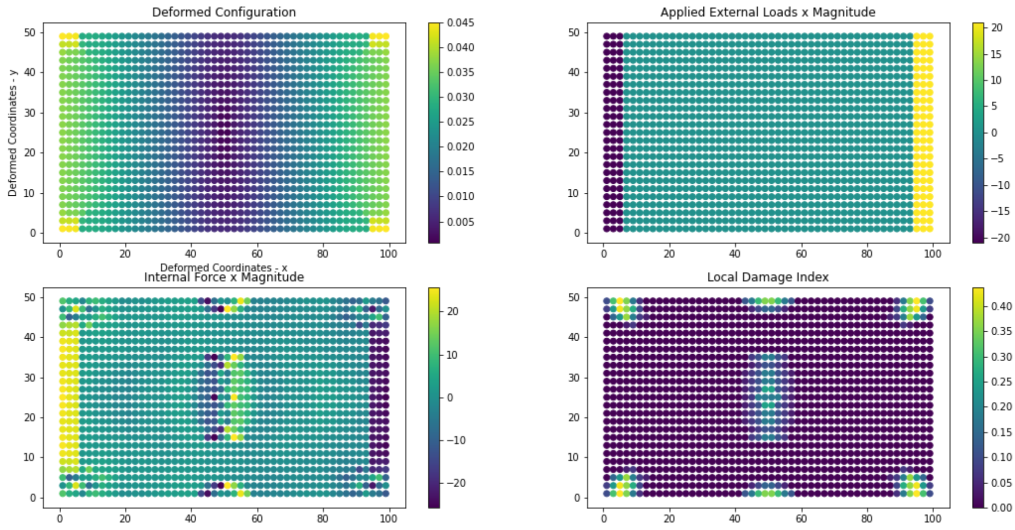
$$\Delta t = \min \sqrt{\frac{2\rho}{\sum_{j=1} C^{eff}_{ij} V_j}} \tag{40}$$

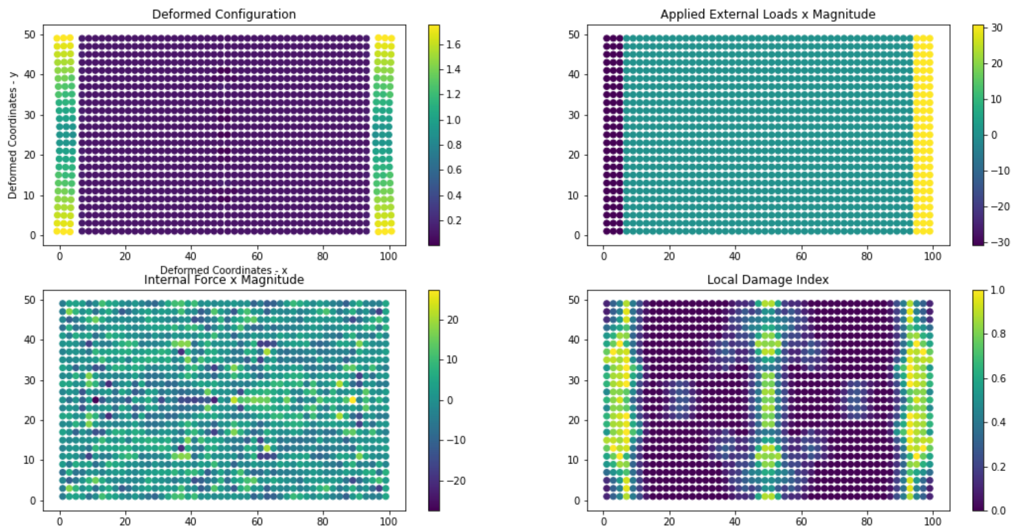
$$C^{eff}_{ij} = \frac{1}{|\vec{\xi}|}\frac{18\kappa}{\pi\delta^4} \tag{41}$$

where an effective modulus C_{ij} is used as suggested. Δt for all material points are calculated and the minimum is selected as the stable step.

4 Example

Uniaxially Loaded Plate: As an example case, a plate example is shown below.





References

[Silling and Lehoucq(2010)] S. Silling and R. Lehoucq, "Peridynamic theory of solid mechanics," in *Advances in Applied Mechanics*, ser. Advances in Applied Mechanics, H. Aref and E. van der Giessen, Eds. Elsevier, 2010, vol. 44, pp. 73–168. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065215610440028

[Madenci and Oterkus(2013)] E. Madenci and E. Oterkus, *Peridynamic Theory and Its Applications*, 06 2013.

[Silling and Askari(2005)] S. Silling and E. Askari, "A meshfree method based on the peridynamic model of solid mechanics," *Computers & Structures*, vol. 83, no. 17, pp. 1526–1535, 2005, advances in Meshfree Methods. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045794905000805

[Littlewood(2015)] D. J. Littlewood, "Roadmap for peridynamic software implementation," 10 2015. [Online]. Available: https://www.osti.gov/biblio/1226115

5 Code:

```
class Body:
    config1=[
        'Deformation',
        'Velocity',
        'Acceleration',
        'InternalForce',
        'ExternalForce'
        ]
    def __init__(self,ConstitutiveModel,ReferenceCoordinates,Volumes,delta):

        self.delta=delta
        self.ReferenceCoordinates=ReferenceCoordinates
        self.Volumes = Volumes
        self.ConstitutiveModel=ConstitutiveModel
        [self.__setattr__(name, zeros(shape=self.ReferenceCoordinates.shape))
            for name in self.config1
            ]
        self.MaterialPointSetup()
        self.MassMatrix = eye(N=len(self.ReferenceCoordinates)) * self.Volumes * self.ConstitutiveModel.Rho#kg
        self.InverseMassMatrix = inv(self.MassMatrix)

    def MaterialPointSetup(self):
        self.MaterialPointList = [
            MaterialPointInABody(label, self)
            for label in range(len(self.ReferenceCoordinates))
            ]

class MaterialPoint:
    def __init__(self,ReferenceCoordinates):
        self.ReferenceCoordinates = ReferenceCoordinates

class MaterialPointInABody:

    def __init__(self,label,Body):
        self.Body = Body
        self.label = label
        self.ReferenceCoordinates = self.Body.ReferenceCoordinates[self.label]
        self.volume = self.Body.Volumes[self.label]
        self.GetNeighbors()
        self.BondDamageScalars = ones(shape=self.NeighborList.shape,dtype=bool)


    def ReferenceRelativePositionVectors(self):
        return self.Body.ReferenceCoordinates[self.NeighborList]-self.ReferenceCoordinates
    def DeformedRelativePositionVectors(self):
```

```python
        return self.ReferenceRelativePositionVectors() + self.Body.Deformation[self.NeighborList]-self.Deformation()
    def StretchScalars(self):
        stretchs=( norm (  self.DeformedRelativePositionVectors(), axis=1 ) - norm( self.ReferenceRelativePositionVectors(),axis=1))/norm(self.ReferenceRelativePositionVectors(),axis=1)
        self.BondDamageScalars = self.BondDamageScalars* (stretchs<self.Body.ConstitutiveModel.sc)
        stretchs = stretchs * (self.BondDamageScalars).astype(int)
        return stretchs
    def LambdaScalars(self):
        return (self.ReferenceRelativePositionVectors() * self.DeformedRelativePositionVectors()).sum(axis=1) / norm(self.ReferenceRelativePositionVectors(),axis=1) / norm(self.DeformedRelativePositionVectors(),axis=1)
    def DilatationScalar(self):
        return self.Body.ConstitutiveModel.d * self.Body.delta * (self.StretchScalars() * self.LambdaScalars() * self.Body.Volumes[self.NeighborList].flatten()).sum()
    def StrainEnergyDensityScalar(self):
        return self.Body.ConstitutiveModel.a * self.DilatationScalar()**2 + self.Body.ConstitutiveModel.b * (self.Body.delta/norm(self.ReferenceRelativePositionVectors(),axis=1) * (norm(self.DeformedRelativePositionVectors(),axis=1)-norm(self.ReferenceRelativePositionVectors(),axis=1))**2*self.Body.Volumes[self.NeighborList]).sum()
    def DeformedRelativeDeformationDirectionVectors(self):
        return self.DeformedRelativePositionVectors()/norm(self.DeformedRelativePositionVectors(),axis=1).reshape((self.DeformedRelativePositionVectors().shape[0],1))
    def InternalForceScalars(self):
        return ( 2 * self.Body.ConstitutiveModel.a * self.Body.ConstitutiveModel.d * self.Body.delta * self.LambdaScalars() * self.DilatationScalar() / norm(self.ReferenceRelativePositionVectors(),axis=1) + 2 * self.Body.ConstitutiveModel.b * self.Body.delta * self.StretchScalars())
    def InternalForceVectorsHost(self):
        return (self.DeformedRelativeDeformationDirectionVectors() * self.InternalForceScalars().reshape((self.InternalForceScalars().shape[0],1)) * self.Body.Volumes[self.NeighborList].reshape((self.Body.Volumes[self.NeighborList].shape[0],1))).sum(axis=0)
    def InternalForceVectorsNeighbors(self):
        return self.DeformedRelativeDeformationDirectionVectors() * self.InternalForceScalars().reshape((self.InternalForceScalars().shape[0],1)) * self.volume
    def UpdateInternalForceVectors(self):
        self.Body.InternalForce[self.label,:] += self.InternalForceVectorsHost()
        self.Body.InternalForce[self.NeighborList,:] -= self.InternalForceVectorsNeighbors()
```

```python

    def GetNeighbors(self):
        self.NeighborList = where(
            (norm(self.ReferenceCoordinates-self.Body.ReferenceCoordinates,axis=1)<=self.Body.delta)*(norm(self.ReferenceCoordinates-self.Body.ReferenceCoordinates,axis=1)!=0.)
            )[0]

    def Deformation(self):
        return self.Body.Deformation[self.label]
    def LocalDamageIndex(self):
        return 1 - (self.Body.Volumes[self.NeighborList].flatten()*self.BondDamageScalars.astype(int)).sum() / (self.Body.Volumes[self.NeighborList]).sum()
class IsotropicMaterial3D:
    def __init__(self,Kappa,Mu,Rho,delta):
        self.Kappa = Kappa
        self.Mu = Mu
        self.Rho = Rho
        self.a = 1 / 2 * ( Kappa - 5 * Mu / 3)
        self.b = 15 * Mu / 2 / pi / delta**5
        self.d = 9 / 4 / pi / delta**4

class IsotropicMaterialPlaneStress:
    def __init__(self,Kappa,Mu,Rho,delta,thickness,Gc):
        self.Kappa = Kappa
        self.Mu = Mu
        self.Rho = Rho
        self.a = 1 / 2 * (Kappa-2*Mu)
        self.b = 6 * Mu / pi / thickness / delta**4
        self.d = 2 / pi/ thickness / delta**3
        self.delta = delta
        self.Gc = Gc
        self.sc = self.critical_stretch()
    def critical_stretch(self):
        return (self.Gc / (6 / pi * self.Mu + 16 / 9 / pi**2 * (self.Kappa - 2*self.Mu))/self.delta)**0.5

class ExplicitIntegrator:
    def __init__(self,Body,max_steps=10000):
        self.Body=Body
        self.Time = 0.0
        self.step=0
        self.max_steps=max_steps
        self.SetDeltaTime()
    def SetDeltaTime(self):
        CTS=1.
        for MP in self.Body.MaterialPointList:
            CpEffs = 18 * self.Body.ConstitutiveModel.Kappa / norm(MP.ReferenceRelativePositionVectors(),axis=1) / pi / self.Body.delta**4
            CTSi = (2*self.Body.ConstitutiveModel.Rho / (CpEffs*self.Body.Volumes[MP.NeighborList]).sum()/1000)**0.5
```

```
            if CTSi<CTS:
                CTS=CTSi
        self.DeltaTime = CTS*0.7

    def integrate(self):
        Time1Step = self.Time+self.DeltaTime
        TimeHalfStep = 0.5*(Time1Step+self.Time)
        VelocityHalfStep = self.Body.Velocity + self.Body.Acceleration*(TimeHalfStep - self.Time)
        self.Body.Deformation += VelocityHalfStep * self.DeltaTime
        self.Body.InternalForce[:,:]=0.
        [MP.UpdateInternalForceVectors() for MP in self.Body.MaterialPointList]
        self.Body.Acceleration = matmul(self.Body.InverseMassMatrix,(self.Body.InternalForce+self.Body.ExternalForce)*self.Body.Volumes)
        self.Body.Velocity = VelocityHalfStep + (Time1Step-TimeHalfStep)*self.Body.Acceleration
        self.Time+=self.DeltaTime
        self.step+=1

def main_func():
    Kappa = 140 * 10**9 #Pascal = N/m2
    Kappa = Kappa *(1/1000)**2#N/mm2
    Mu = 80 * 10**9 #Pascal = N/m2
    Mu = Mu *(1/1000)**2#N/mm2
    Rho = 8050#kg/m3
    Rho=Rho*(1/1000)**3#kg/mm3
    nu = (1-Mu/Kappa)/(1+Mu/Kappa)
    E = Kappa * 2 * (1-nu)
    KIc = 12 #MPa*m = N/mm2 * m**1/2
    GIc = KIc**2*1000/E # N/mm
    L=100.
    W=L/2.
    t = 3.
    dx = 0.5*4
    delta = dx*3.1
    metal = IsotropicMaterialPlaneStress(Kappa,Mu,Rho,delta,t,GIc)
    Plate=array([[x,y] for x in arange(dx/2,L+dx/2,dx) for y in arange(dx/2,W+dx/2,dx)])
    Volumes = array([[L*W*t/len(Plate)] for i in range(len(Plate))])
    example=Body(metal, Plate, Volumes,delta)
    Left_Application = where((Plate[:,0]>=L-3*dx))[0]
    Right_Application = where((Plate[:,0]<=3*dx))[0]
    Total_Applied_Force = 1*W*t
    example.ExternalForce[Left_Application,0] = Total_Applied_Force/(len(Left_Application))
    example.ExternalForce[Right_Application,0] = -Total_Applied_Force/(len(Right_Application))
    integrator = ExplicitIntegrator(example)
    return example,integrator,Left_Application,Right_Application,Total_Applied_Force
if __name__=='__main__':
```

```
    example,integrator,Left_Application,Right_Application,Total_Applied_Force=main_func()
    from matplotlib import pyplot as plt
    import pickle
    while integrator.step<1500:
        integrator.integrate()
        if not integrator.step%250:
            print(integrator.step)
            fig,ax = plt.subplots(2,2)
            fig.set_size_inches(18,9)
            ax1 = ax[0][0]
            ax1.set_autoscale_on(False)
            ax1.set_xlim(-5,105)
            ax1.set_ylim(-2.5,52.5)
            im1=ax1.scatter(example.ReferenceCoordinates[:,0] + example.Deformation[:,0],example.ReferenceCoordinates[:,1] + example.Deformation[:,1],c=norm(example.Deformation,axis=1))
            fig.colorbar(im1,ax=ax1)
            ax1.set_title('Deformed Configuration')
            ax1.set_xlabel('Deformed Coordinates - x')
            ax1.set_ylabel('Deformed Coordinates - y')
            ax2=ax[0][1]
            ax2.set_xlim(-5,105)
            ax2.set_ylim(-2.5,52.5)
            ax2.set_autoscale_on(False)
            im2=ax2.scatter(example.ReferenceCoordinates[:,0],example.ReferenceCoordinates[:,1],c=example.ExternalForce[:,0])
            fig.colorbar(im2,ax=ax2)
            ax2.set_title('Applied External Loads x Magnitude')
            ax3=ax[1][0]
            ax3.set_xlim(-5,105)
            ax3.set_ylim(-2.5,52.5)
            ax3.set_autoscale_on(False)
            im3=ax3.scatter(example.ReferenceCoordinates[:,0],example.ReferenceCoordinates[:,1],c=example.InternalForce[:,0])
            fig.colorbar(im3,ax=ax3)
            ax3.set_title('Internal Force x Magnitude')
            ax4=ax[1][1]
            ax4.set_xlim(-5,105)
            ax4.set_ylim(-2.5,52.5)
            ax4.set_autoscale_on(False)
            im4=ax4.scatter(example.ReferenceCoordinates[:,0],example.ReferenceCoordinates[:,1],c=[i.LocalDamageIndex() for i in example.MaterialPointList])
            fig.colorbar(im4,ax=ax4)
            im4.colorbar.vmax=1.0
            im4.colorbar.vmin=0.0
            ax4.set_title('Local Damage Index')
            fig
            fig.savefig(f'Madenci_Oterkus_Plate_Example_v4_wFailure{integrator.step}')
            pickle.dump([example,integrator],open(f'Madenci_Oterkus_Plate_Example_v4_wFailure{integrator.step}','wb'))
```

```
            example.ExternalForce[Left_Application,0] +=
            ↪  Total_Applied_Force/(len(Left_Application))
            example.ExternalForce[Right_Application,0] -=
            ↪  Total_Applied_Force/(len(Right_Application))
```